



SUPRA SERVER PDM

RDM Administration Guide
(OS/390 & VSE)

P26-8220-64




SUPRA® Server PDM RDM Administration Guide (OS/390 & VSE)

Publication Number P26-8220-64

© 1985–1989, 1991–1994, 1997, 1998, 2000, 2002 Cincom Systems, Inc.
All rights reserved

This document contains unpublished, confidential, and proprietary information of Cincom. No disclosure or use of any portion of the contents of these materials may be made without the express written consent of Cincom.

The following are trademarks, registered trademarks, or service marks of Cincom Systems, Inc.:

AD/Advantage®	iD CinDoc™	MANTIS®
C+A-RE™	iD CinDoc Web™	Socrates®
CINCOM®	iD Consulting™	Socrates® XML
Cincom Encompass®	iD Correspondence™	SPECTRA™
Cincom Smalltalk™	iD Correspondence Express™	SUPRA®
Cincom SupportWeb®	iD Environment™	SUPRA® Server
CINCOM SYSTEMS®	iD Solutions™	Visual Smalltalk®
 gOOi™	intelligent Document Solutions™	VisualWorks®
	Intermax™	

UniSQL™ is a trademark of UniSQL, Inc.
ObjectStudio® is a registered trademark of CinMark Systems, Inc.

All other trademarks are trademarks or registered trademarks of their respective companies.

Cincom Systems, Inc.
55 Merchant Street
Cincinnati, Ohio 45246-3732
U.S.A.

PHONE: (513) 612-2300
FAX: (513) 612-2000
WORLD WIDE WEB: <http://www.cincom.com>

Attention:

Some Cincom products, programs, or services referred to in this publication may not be available in all countries in which Cincom does business. Additionally, some Cincom products, programs, or services may not be available for all operating systems or all product releases. Contact your Cincom representative to be certain the items are available to you.

Release information for this manual

The *SUPRA Server PDM RDM Administration Guide (OS/390 & VSE)*, P26-8220-64, is dated January 15, 2002. This document supports Release 2.7 of SUPRA Server PDM in IBM mainframe environments.

We welcome your comments

We encourage critiques concerning the technical content and organization of this manual. Please take the [survey](#) provided with the online documentation at your convenience.

Cincom Technical Support for SUPRA Server PDM

FAX: (513) 612-2000
Attn: SUPRA Server Support

E-mail: helpna@cincom.com

Phone: 1-800-727-3525

Mail: Cincom Systems, Inc.
Attn: SUPRA Server Support
55 Merchant Street
Cincinnati, OH 45246-3732
U.S.A.

Contents

About this book	xi
Using this document.....	xi
Document organization	xi
Revisions to this manual	xii
Conventions	xiii
SUPRA Server documentation series	xvi
 Overview of the SUPRA Relational Data Manager	 19
The user's perspective—views	21
Accessing views	21
Subsetting views	22
The DBAs perspective—three-schema architecture.....	23
Internal schema.....	23
Conceptual schema—base views	24
External schema—derived views	25
RDM security.....	26
The DBA function in the RDM	27
System requirements	28
Directory	28
Physical Data Manager	28
Hardware.....	28

Accessing user data	29
Overview of base views and derived views	29
Designing derived views	31
How the RDM constructs rows	32
Providing keyed access	33
Unique keys	35
Nonunique keys	38
Constant keys	39
Required columns	39
Using domains, null values, and default values for physical fields	40
Validation options	41
Null values	45
Default values	47
 Modifying user data	 49
Changing the database contents	49
Inserting information to the database	51
Updating information on the database	52
Deleting information from the database	53
Allowing shared column values	54
Retrieving data with the RDM	55
Database penetration	56
Database sweep	57
Indexing	57
Navigational constraints and boundaries	57
Status indicators	58
Function status indicators	59
Column status indicators	60
Validity status indicators	62
 Defining and using derived views	 63
Defining derived views	64
Column definition	64
Access definition	72
Examples of derived view definitions	76
Base relations	76
Base views	78
Derived views	80
Processing derived views	85
Processing the GET command	85
Processing the INSERT command	86

Maintaining referential integrity	87
Integrity rules and checking.....	89
Foreign key value integrity.....	90
Insertion integrity.....	91
Update integrity.....	93
GET processing	95
Deletion integrity.....	96
Referential integrity examples.....	99
 Maintaining the RDM	 103
Defining and testing views with DBAID	104
Signing on to DBAID and RDM	105
Defining base views	106
Defining a derived view	110
Retrieving records.....	111
Inserting records	112
Updating a row	113
Modifying a view definition	114
Maintaining current programs and views	115
Checking currentness of program.....	120
Checking currentness of view bindings.....	120
Optimizing performance	121
Global view support.....	122
View binding.....	125
Installing the RDM resident module in shared memory	126
Gathering and interpreting statistics.....	128
Gathering statistics with DBAID	128
Gathering statistics in an application program	128
Interpreting RDM statistics	129
Statistics example	129
Relating views to users	130
Recovering data	132

Managing views with the DBAID commands	133
Introduction to DBAID	133
System commands	136
Editing commands	136
RDML commands	137
Built-in view commands	138
Statistic commands	138
= command	139
BIND command	140
BYE command	142
BY-LEVEL command	143
CAUTIOUS Command	145
COLUMN-DEFN command	146
COLUMN-TEXT command	148
COMMIT command	149
COPY command	150
DEFINE command	152
DELETE command	153
DENY command	155
EDIT command	156
ERASE command	157
FIELD-DEFN command	158
FORGET command	160
GET command	161
GO command	164
INSERT command	168
KEEP command	172
Line-number command	173
LINESIZE command	175
LIST command	176
MARK command	178
MARKS command	179
OPEN command	180
PAGESIZE command	182
PERMIT command	183
PRINT-STATS command	184
PUBLIC-DENY command	185
PUBLIC-PERMIT command	186
PUBLIC-VIEWS command	187
RELEASE command	188
REMOVE command	189
RENUMBER command	191
RESET command	192
SAVE command	193
SHOW-NAVIGATION command	195
SIGN-OFF command	197

SIGN-ON command.....	198
STATS command.....	199
STATS-OFF command	200
STATS-ON command	201
SURE command	202
UNDEFINE command.....	203
UPDATE command.....	204
USER-LIST command	207
VIEW-DEFN command.....	208
VIEWS command.....	209
VIEWS-FOR-USER command	210
Using the RDM reports	211
DBA report.....	213
Programmer's report	215
End user report	218
Impact of change report	220
Files impacting views report.....	220
Views impacting views report.....	222
Views impacting programs report	223
Views used by programs report	224
Configuring the RDM for your environment	225
Overview of configuring the RDM for your environment	225
Configuring the RDM XA storage.....	228
Interaction of options parameters.....	229
Environment description parameters	230
The connect/sinon process	231
OPER CONNECT parameters.....	232
RDML processing.....	233
PDM thread processing.....	235
CICS processing	236
Customizing the RDM processing with user exits	237
Overview of customizing the RDM processing with user exits.....	237
Using database exits.....	240
Using environment-independent database exits	242
Using environment-dependent database exits.....	245
Using RDML exits	249
Using the before-function exit (CSVXBFOR)	250
Using the after-function exit (CSVXAFTF).....	252
Using the TASKID exit (CSVXTSID).....	254
Using validation exits.....	256

Setting the online RDM options with macros **261**

 Overview of setting the online RDM options with macros..... 261

Index **273**

About this book

Using this document

This manual is intended for the DBA, the person responsible for designing and modifying the logical and physical structure of your database.

Document organization

The information in this manual is organized as follows:

Chapter 1—Overview of the SUPRA Relational Data Manager

Provides an overview of the RDM.

Chapter 2—Accessing user data

Explains how to design derived views, how the RDM constructs a row of data, and how to specify keys, null values, defaults, and validation options for a view.

Chapter 3—Modifying user data

Explains how to modify user data.

Chapter 4—Defining and using derived views

Explains how to define and use derived views.

Chapter 5—Maintaining referential integrity

Explains referential integrity and how to maintain it.

Chapter 6—Maintaining the RDM

Describes the RDM maintenance functions. It describes how to create and test views, how to create and maintain the relationship of views with applications and users, how to analyze and optimize the RDM performance, and how to recover data.

Chapter 7—Managing views with the DBAID commands

Lists and describes the DBAID utility commands for managing views.

Chapter 8—Using the RDM reports

Explains how to use the RDM reports.

Chapter 9—Configuring the RDM for your environment

Tells how to find the information to configure the RDM for your operating environment. It also summarizes the new RDM features.

Appendix A—Customizing the RDM processing with user exits

Describes the RDM user exits for customizing RDM processing.

Appendix B—Setting the online RDM options with macros

Describes the RDM macro C\$VOOPTM for setting user options.

Index

Revisions to this manual

The following changes have been made for this release:

- ◆ The illustration under “**Using the after-function exit (CSVXAFTR)**” on page 252 has been corrected to show 5 parameters.
- ◆ The table showing **addressing modes for RDM user exit programs** on page 240 has been corrected to show the addressing modes for modules CSVXCFNC and CSVXCSTA are the same as the addressing mode for the invoking user application.
- ◆ The NORMAL product is no longer distributed. If you use NORMAL, retain your files and previous documentation. References to NORMAL in this document have been deleted.

Conventions

The following table describes the conventions used in this document series:

Convention	Description	Example
Constant width type	Represents screen images and segments of code.	<pre>PUT 'customer.dat' GET 'miller\customer.dat' PUT '\DEV\RMT0'</pre>
Slashed b (<i>b</i>)	Indicates a space (blank). The example indicates that four spaces appear between the keywords.	<pre>BEGNbbbbSERIAL</pre>
Brackets []	Indicate optional selection of parameters. (Do not attempt to enter brackets or to stack parameters.) Brackets indicate one of the following situations:	
	A single item enclosed by brackets indicates that the item is optional and can be omitted.	<pre>[WHERE <i>search-condition</i>]</pre>
	The example indicates that you can optionally enter a WHERE clause.	
	Stacked items enclosed by brackets represent optional alternatives, one of which can be selected.	<pre>[<u>(WAIT)</u> (NOWAIT)]</pre>
	The example indicates that you can optionally enter either WAIT or NOWAIT. (WAIT is underlined to signify that it is the default.)	

Convention	Description	Example
Braces { }	<p>Indicate selection of parameters. (Do not attempt to enter braces or to stack parameters.) Braces surrounding stacked items represent alternatives, one of which you must select.</p> <p>The example indicates that you must enter ON or OFF when using the MONITOR statement.</p>	<p>MONITOR {ON } {OFF }</p>
	<p>Underlining also indicates an allowable abbreviation or the shortest truncation allowed.</p> <p>The example indicates that you can enter either STAT or STATISTICS.</p>	<p><u>STATISTICS</u></p>
(For new information)	<p>Technical changes and new information pertinent to this release are marked by underlining.</p>	<p>The minimum record length is 21 for primary datasets and 41 for related datasets.</p>
Ellipsis points...	<p>Indicate that the preceding item can be repeated.</p> <p>The example indicates that you can enter multiple host variables and associated indicator variables.</p>	<p>INTO :host-variable [:ind-variable],...</p>

Convention	Description	Example
UPPERCASE lowercase	In most operating environments, keywords are not case-sensitive, and they are represented in uppercase. You can enter them in either uppercase or lowercase.	<code>COPY MY_DATA.SEQ</code> <code>HOLD_DATA.SEQ</code>
<i>Italics</i>	Indicate variables you replace with a value, a column name, a file name, and so on. The example indicates that you must substitute the name of a table.	<code>FROM table-name</code>
Punctuation marks	Indicate required syntax that you must code exactly as presented. () parentheses . period , comma : colon ' ' single quotation marks	<code>(user-id, password, db-name)</code> <code>INFILE 'Cust.Memo' CONTROL</code> <code>LEN4</code>
SMALL CAPS	Represent a keystroke. Multiple keystrokes are hyphenated.	ALT-TAB
OS/390 VSE	Information specific to a certain operating system is flagged by a symbol in a shadowed box (OS/390) indicating which operating system is being discussed. Skip any information that does not pertain to your environment.	OS/390 See the SUPRA Server procedure library member TIS\$RDM for a list of RDM procedures. VSE See the SUPRA Server RDM sublibrary member TXJ\$INDX for a list of JCL.

SUPRA Server documentation series

SUPRA Server is the advanced relational database management system for high-volume, update-oriented production processing. A number of tools are available with SUPRA Server including Directory Maintenance, DBA utilities, DBAID, SPECTRA, and MANTIS. The following list shows the manuals and tools used to fulfill the data management and retrieval requirements for various tasks. Some of these tools are optional. Therefore, you may not have all the manuals listed. For a brief synopsis of each manual, refer to the *SUPRA Server PDM Digest (OS/390 & VSE)*, P26-9062.

Overview

- ◆ *SUPRA Server PDM Digest (OS/390 & VSE)*, P26-9062

Getting started

- ◆ *SUPRA Server PDM Migration Guide (OS/390 & VSE)*, P26-0550*
- ◆ *SUPRA Server PDM CICS Connector Systems Programming Guide (OS/390 & VSE)*, P26-7452

General use

- ◆ *SUPRA Server PDM Glossary*, P26-0675
- ◆ *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126

Database administration tasks

- ◆ *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250
- ◆ *SUPRA Server PDM Directory Online User's Guide (OS/390 & VSE)*, P26-1260
- ◆ *SUPRA Server PDM Directory Batch User's Guide (OS/390 & VSE)*, P26-1261
- ◆ *SUPRA Server PDM DBA Utilities User's Guide (OS/390 & VSE)*, P26-6260
- ◆ *SUPRA Server PDM Logging and Recovery (OS/390 & VSE)*, P26-2223
- ◆ *SUPRA Server PDM Tuning Guide (OS/390 & VSE)*, P26-0225
- ◆ *SUPRA Server PDM RDM Administration Guide (OS/390 & VSE)*, P26-8220
- ◆ *SUPRA Server PDM RDM PDM Support Supplement (OS/390 & VSE)*, P26-8221
- ◆ *SUPRA Server PDM RDM VSAM Support Supplement (OS/390 & VSE)*, P26-8222
- ◆ *SUPRA Server PDM Migration Guide (OS/390 & VSE)*, P26-0550*
- ◆ *SUPRA Server PDM Windows Client Support User's Guide*, P26-7500*
- ◆ *SPECTRA Administrator's Guide*, P26-9220

Application programming tasks

- ◆ *SUPRA Server PDM DML Programming Guide (OS/390 & VSE)*, P26-4340
- ◆ *SUPRA Server PDM RDM COBOL Programming Guide (OS/390 & VSE)*, P26-8330
- ◆ *SUPRA Server PDM RDM PL/1 Programming Guide (OS/390 & VSE)*, P26-8331
- ◆ *SUPRA Server PDM Migration Guide (OS/390 & VSE)*, P26-0550*
- ◆ *SUPRA Server PDM Windows Client Support User's Guide*, P26-7500*

Report tasks

- ◆ *SPECTRA User's Guide*, P26-9561



Manuals marked with an asterisk (*) are listed more than once because you use them for multiple tasks.



Educational material is available from your regional Cincom education department.

1

Overview of the SUPRA Relational Data Manager

The SUPRA Relational Data Manager (RDM) provides a relational view of data for end users and application programs. The RDM insulates end users and application programs from the physical structure of the databases and from changes in that structure.

The RDM does the following:

- ◆ Isolates application programmers and end users from the physical database implementation. Allows the DBA to restructure the database without requiring programs to be rewritten or recompiled.
- ◆ Provides programmers with a simplified Relational Data Manipulation Language (RDML) for retrieving and modifying the database contents. Application programs with RDML tend to be smaller and take less time to code.
- ◆ Allows the DBA to control database security by specifying the levels of access allowed to specified users through specified views.
- ◆ Enforces database integrity.
- ◆ Maintains data in a tabular (relational) structure.
- ◆ Supports relational operators.
- ◆ Processes data as relations.
- ◆ Supports multiple physical file structures.

- ◆ Supports null values.
- ◆ Performs automatic data validation.
- ◆ Supports default values for physical fields. People in your organization would have different perspectives on SUPRA Server relational data management depending on their job function:
- ◆ The DBA uses DBAID or Directory Maintenance to create base views (views that access files). The DBA also uses DBAID or Directory Maintenance to create derived views (views that access base views or other derived views).
- ◆ The application programmer uses DBAID to create derived views. The application programmer also creates and runs programs in COBOL or PL/1 that access data with views, and may use SPECTRA and/or MANTIS to access data with views.
- ◆ The end user uses SPECTRA, MANTIS, and/or in-house application programs to access data with views.

The user's perspective—views

Using RDM, the application programmer or end user can access database information without needing to know the data's physical location, physical structure, or integrity constraints. RDM allows the user to view data as if it were arranged in tables or relations, consisting of rows and columns as shown in the following illustration:

VIEW - A Table of Data			
CUSTOMER Number	CUSTOMER Name	CUSTOMER Class	
E40000	DOUG REED	Q1	ROW
F80081	TOM LANGDON	B4	ROW
H22233	ATHENS INC	J1	ROW

COLUMN
 COLUMN
 COLUMN

Accessing views

The user accesses views provided by the DBA. The DBA creates views and relates them to users. The DBA defines the maintenance action (**INSERT**, **UPDATE**, and **DELETE**) that users can perform with a view. The DBA can also limit a view to read-only access, with no maintenance capabilities.

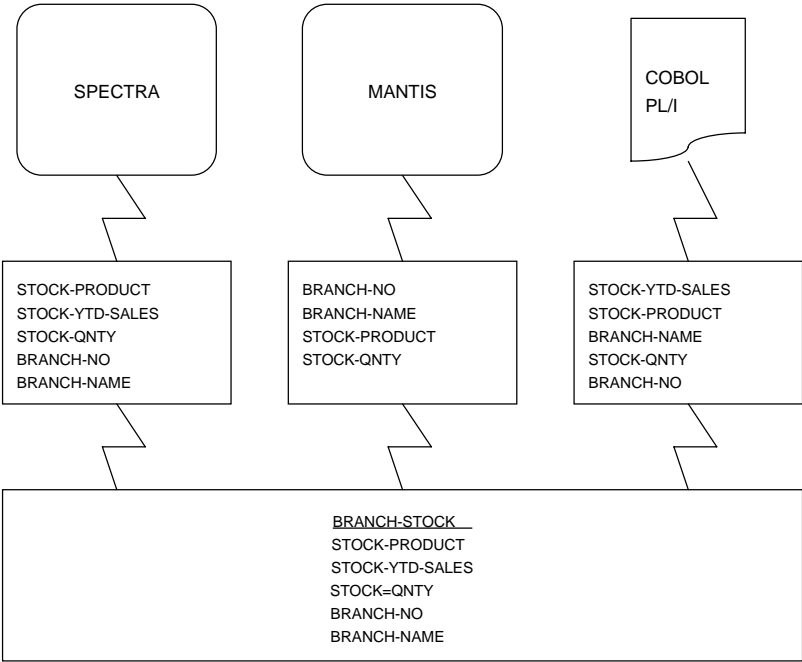
Application programmers access views through program logic using COBOL, PL/1, or MANTIS. COBOL and PL/1 programmers use the Relational Data Manipulation Language (RDML) precompilers to compile RDML statements into executable code. Refer to the *SUPRA Server PDM RDM COBOL Programming Guide (OS/390 & VSE)*, P26-8330, or the *SUPRA Server PDM RDM PL/1 Programming Guide (OS/390 & VSE)*, P26-8331, for information on using RDML statements in application programs.

End users can access views directly through SPECTRA, a relational query and update tool.

Subsetting views

The DBA can define generalized views that many users and programs can access. The user can create a subset of a view's columns or reorder the columns to meet a specific need. A subset of a view is called a user view.

The following figure shows an example of different users subsetting and reordering a view. Each user is accessing the **BRANCH-STOCK** view which contains five columns. The **SPECTRA** user uses the entire view in the same sequence as specified in the View Definition. The **MANTIS** application uses only part of the available data, and also reorders the columns. The application programs on the right use all of the columns, but reorder them.

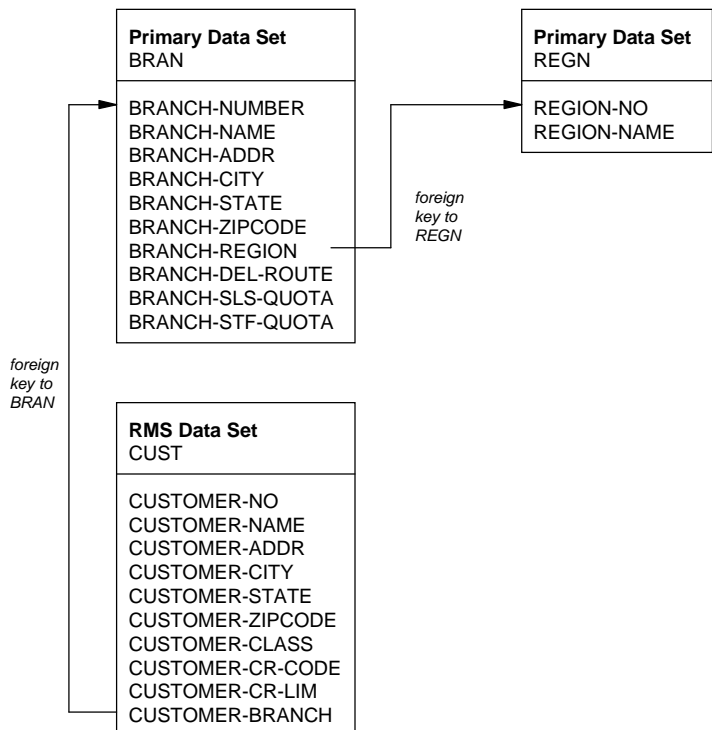


The DBA's perspective—three-schema architecture

The DBA designs and maintains the database. SUPRA Server provides the DBA with a three-schema architecture, allowing a physical and logical implementation that is insulated from change. Three-schema architecture consists of internal, conceptual, and external schemas.

Internal schema

The internal schema is at the lowest level of three-schema architecture. The internal schema defines the physical contents of the files. RDM supports PDM primary files, PDM related files, and KSDS VSAM files. The internal schema by itself constitutes a one-schema architecture, as shown in the following figure:

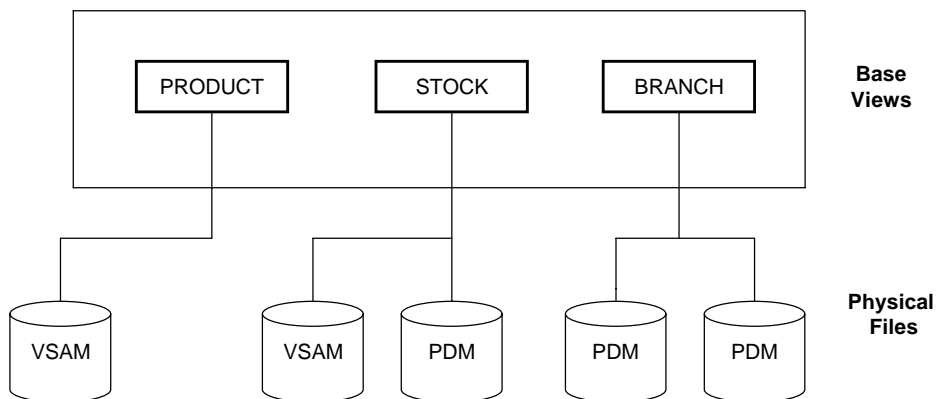


Conceptual schema—base views

The conceptual schema is at the middle level of the three-schema architecture. The conceptual and internal schemas, without the external schema, constitute a two-schema architecture (see the following illustration). The conceptual schema defines logical access to the physical database.

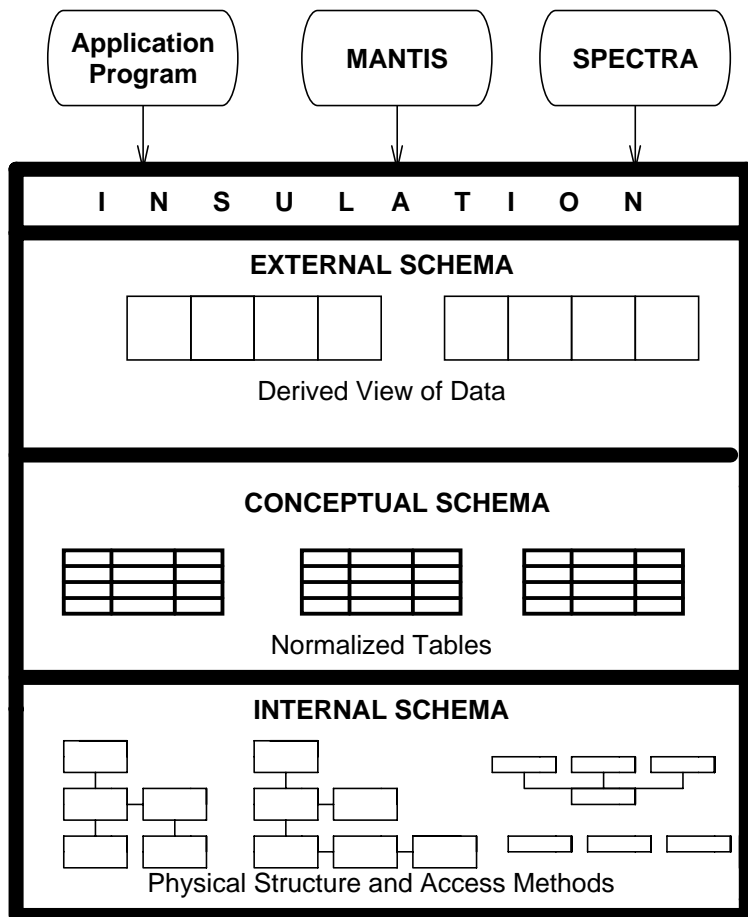
The conceptual schema consists of base views. Base views access physical files directly. Referential integrity and file security are handled best at the base view level.

You can define base views using the DBAID utility or Directory Maintenance. The DBAID utility is an online and batch tool for defining and testing views, and for relating them to users. See “[Maintaining the RDM](#)” on page 103 for more information on using the DBAID utility. Refer to the *SUPRA Server PDM RDM PDM Support Supplement (OS/390 & VSE)*, P26-8221, or the *SUPRA Server PDM RDM VSAM Support Supplement (OS/390 & VSE)*, P26-8222, for information on creating base views based on the physical file type.



External schema—derived views

At the uppermost level of the three-schema architecture is the external schema consisting of derived views. The external schema represents the database independent of logical and physical structures, integrity, and physical access. The following figure illustrates the three-schema architecture:



Derived views are the implementation of the external schema. The application programmer or SPECTRA user sees no difference between base or derived views. They are both “views” of data. You can place more restrictive security or higher levels of security on derived views; however, you cannot override any file security or integrity specified in the base view. Base views insulate derived views from changes to the physical database.

You define derived views using the DBAID utility or Directory Maintenance. You store these view definitions on the Directory for application programmers and end users to use later. You also use the DBAID utility to prototype and test new views before putting these views into production. Once you define the views and relate them to users, they are available for use. You can use RDM reports to show the definition of a view, which users can access the view, and which programs use the view. See “[Using the RDM reports](#)” on page 211 for more information about RDM Reports.

RDM security

The RDM controls security in the following ways:

- ◆ User-to-view Relationships define which views the user can use. You can relate both base and derived views to users using the DBAID utility or Directory Maintenance. See “[Relating views to users](#)” on page 130 for information on relating views to users.
- ◆ View Accesses define what actions a view can perform, such as update, delete, and insert by specifying the ALLOW clause in the View Definition. Views can be read-only, insert-only, or unlimited in their ability to perform maintenance actions on the database. The DBA can create derived views that impose additional security restrictions.

The DBA function in the RDM

In an environment where a database includes data shared by many users and programs, the DBA must develop the database definition centrally. In such a shared environment, the database design must meet the needs of the various users. The DBA makes the decisions for how data is to be shared centrally.

The DBA's responsibilities may be spread among the user groups, or they may belong to a central person or staff. Whether you have a database administrator or a database administration group depends on the size and needs of your particular organization. The proper administration of RDM requires broad knowledge of data use throughout the organization. The DBA's function is to do the following:

- ◆ Describe the logical and physical data attributes.
- ◆ Define relationships that exist between data units.
- ◆ Define how to access the data.
- ◆ Provide security, integrity, and validation constraints for the database.
- ◆ Optimize system performance.
- ◆ Maintain control over the definition and generation of data views.
- ◆ Gather users' data needs and define views to fit those needs. The DBA can also assist users in determining the best method for structuring their views and application programs.
- ◆ Control changes made to the View Definitions, and provide copies of definitions and changes to those who need them.

RDM reports are tools for both the DBA and the application programmer. The reports show available views and information about the Directory. See [“Using the RDM reports”](#) on page 211 for information about RDM reports.

With DBAID, the DBA can design, test, and examine the performance of a view before placing it into production use and assigning it to users. See [“Managing views with the DBAID commands”](#) on page 133 for information about using DBAID.

System requirements

You must meet several requirements before RDM can be installed and made operational. For information about running DBAID and RDM in the various operating environments and modes, refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250.

The following sections describe general requirements for RDM.

Directory

The RDM uses the Directory to store view information. The RDML compiler, RDM, and DBAID use the Directory. The RDM run time processors require information contained in the Directory.

The RDM ignores some Directory data, including the following:

- ◆ Views data:
 - Generalized-updates indicator
 - RDM indicator
 - Site-table name
- ◆ User-to-view relationship data:
 - Define-generalized-updates option
 - Execute-generalized-updates option
 - Define-RDM-applications option
 - Execute-RDM-applications option
- ◆ View-to-external-field relationship data:
 - Alias name
 - Record code
 - Control-key indicator

Physical Data Manager

The SUPRA Physical Data Manager (PDM) is required to access the Directory to retrieve and update the information required by the RDM. The PDM is required to access PDM user files. VSAM is required to access KSDS VSAM user files.

Hardware

The RDM runs on an IBM 370 or similar, later-model mainframe CPU (30xx, 43xx, etc.). When running DBAID in an online environment, IBM 3270s (or equivalent equipment) are the only terminal types supported.

2

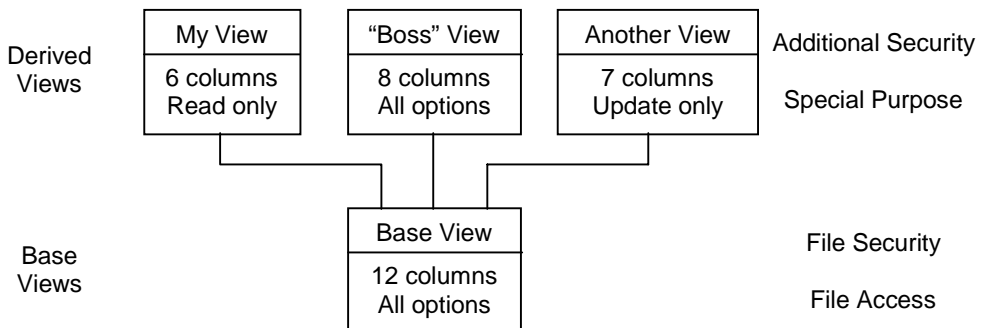
Accessing user data

Overview of base views and derived views

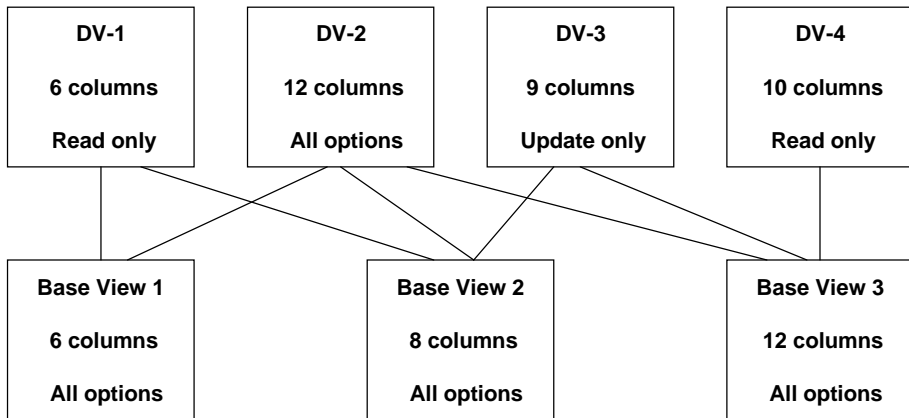
An application programmer or an end user (a SPECTRA user) can access data with two types of views: base views and derived views. Base views access only files. Derived views access base views or other derived views. The difference is not significant to an application programmer or end user; both base views and derived views are seen as tables of data. The difference is important to the DBA who constructs the views.

You can define base views on the Directory using DBAID or Directory Maintenance. The view definition consists of defining columns in a view and specifying the files to access. The syntax for base view definitions is described in the *SUPRA Server PDM RDM PDM Support Supplement (OS/390 & VSE)*, P26-8221, and the *SUPRA Server PDM RDM VSAM Support Supplement (OS/390 & VSE)*, P26-8222. Base view definitions vary depending on the types of files you are accessing.

You build derived views using DBAID. You define the columns that make up the view and which base views to access to gather the columns. Many derived views can access a single base view. Each derived view can reorder the columns or include or exclude different columns. The following figure shows how several derived views can access a single base view. Integrity and view-to-file security are implemented at the base view level; additional security can be implemented in the derived views.



A derived view may access more than one base view. The following figure shows a derived view accessing several base views. Derived view 1 (DV-1) accesses base views 1 and 2, but can read only. Derived view 2 (DV-2) accesses all three base views and can perform all functions (**INSERT**, **UPDATE**, **DELETE**, and **READ**). Derived view 3 (DV-3) has **UPDATE** capabilities and is accessing base views 2 and 3. Derived view 4 (DV-4) is accessing only base view 3, and can only read the view.



Designing derived views

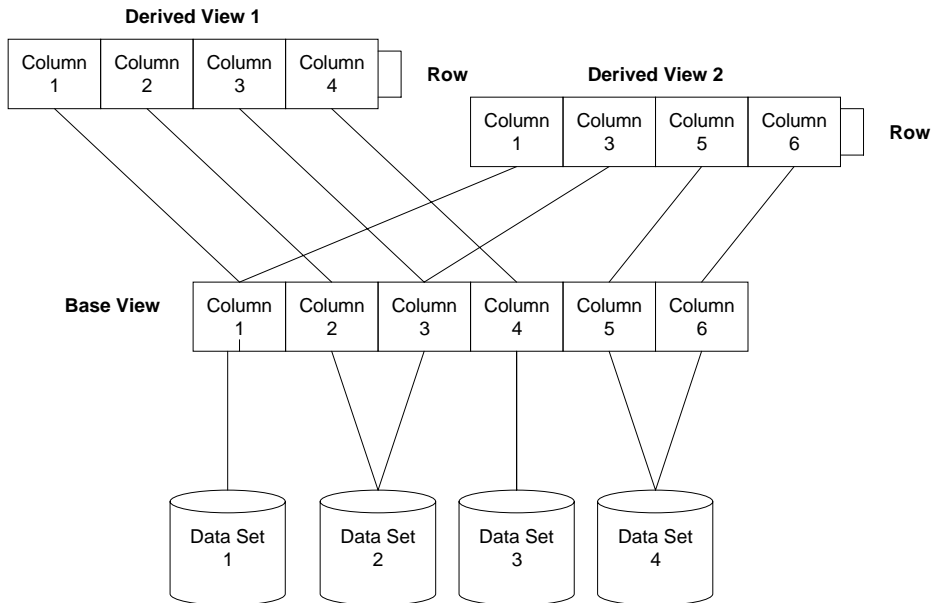
To design derived views, you must be familiar with your base views. You can use Directory Reports to report on the base views, and use DBAID to test and examine them. You can also use SPECTRA to access base views as an end user.

Follow these guidelines to define derived views:

- ◆ Design the derived view for ease of use. Cincom recommends that you limit the size of your views. In general, it is better to have many small or medium size views than it is to have one large view containing all the data.
- ◆ Create customized views according to the various job functions within your organization. Another approach is to have certain views for maintenance, different views for retrievals, and others for reporting.
- ◆ Use the DBAID utility to build and test the views and ensure their validity before placing them into production.
- ◆ Derived views can access multiple base views, but they cannot directly access files.
- ◆ You may impose additional maintenance restrictions (**INSERT**, **UPDATE**, and **DELETE**) on derived views, but you cannot override maintenance restrictions imposed by the base view. Refer to RDM reports or DBAID to examine the maintenance restrictions imposed by a base view.
- ◆ To centralize integrity constraints, specify all referential integrity in the base views.
- ◆ When you construct a view for update, the columns in the row must depend on the logical key value. This helps avoid update anomalies.
- ◆ Every column in a derived view inherits the validation criteria of the underlying base view.
- ◆ It is important for programmers to know if a column is required. “**Required columns**” on page 39 for information on required columns.

How the RDM constructs rows

The RDM constructs one or more rows based upon the view definition that you supply. RDM does this by obtaining data from the files or views named in the ACCESS statements. After RDM obtains the data, it moves the data into each column of the row from the appropriate source field or source column, as shown in the following figure:



Providing keyed access

Through the view definition, you can provide keyed access to data. If you do not provide keyed access, a serial access of the file or view results. Even if a physical key read can be performed on the database file, you can still define a nonkeyed view which would limit that file to sequential access. To be selected, the physical data must be equal to the logical key value.

There is an important difference between defining a logical key and an access key. You define the logical key using the keyword KEY (or NONUNIQUE KEY) in the appropriate column definition(s) in the view definition. You define the access keys in the appropriate access definition(s) in the view definition. The access keys determine the accessing method to use. “[Defining derived views](#)” on page 64 discusses view definitions in detail. In the following example, CUSTOMER-NO and CUSTOMER-NAME are both identified as logical keys. However, CUSTOMER-NO is the access key to the customer file.

```
KEY CUSTOMER-NO
KEY CUSTOMER-NAME
CUSTOMER-ADDR
CUSTOMER-STATE
ACCESS CUST WHERE CUSTOMER-NO = CUSTOMER-NO ALLOW ALL
```

Every view can have zero to nine logical key columns, and the program can supply any number of these key values for the view. A logical key in the view does not, of itself, cause the RDM to perform a physical keyed access. If you define a column as a logical key and it maps to an access key which maps to a physical key (a control key on a PDM primary file), and the user program requests a read and supplies that logical key, the RDM performs a keyed access of the physical file. The RDM goes directly to the requested record.

This does not mean that you can assign only logical keys to access key columns. For example, when you have customer numbers and order numbers, if you define customer number (which would probably be a physical key) as a logical key, the random access of the requested customer number would be very quick. You could just as well define customer name, which is a data field in the file record, as a logical key. In this case, the RDM would service the request, and would sequentially search the file (unless you also had an index on customer name) for a match on the customer name supplied by the program. This is a valid use of the logical key, but it would result in much slower processing because it requires a serial scan of the file.

The REQ option of the column definition designates required columns. If a column is required, it must be present and valid for the RDM to return a row. Otherwise, the row is skipped and not returned.

Each logical key consists of one or more columns. You can assign fixed values (constants) to a key column to constrain the application program to retrieve or update selected records. Logical key columns are required columns. See “[Required columns](#)” on page 39 for an example of the impact of required columns.

You can define four different types of logical key columns: unique key, nonunique key, constant, and unique constant. You can specify logical keys as unique or nonunique depending upon your application requirements and record organization. The following sections provide information on logical key columns and required columns.

Unique keys

A relation with a unique key has one row for each key value. Each row can map to one or more physical files. Therefore, using a unique key with unnormalized views may retrieve more than one row for each unique key.

You can build views that have only unique keys but still return several rows per unique key combination. This occurs when the unique logical keys do not uniquely specify a single logical row. Using the customer order view (from the illustration under “[Compound unique keys](#)” on page 37) as an example, if the customer number column is designated as a unique logical key but the order number column is not, the user would be able to specify customer number on a [GET](#) and retrieve multiple records for each customer number. Basically, this is a generic search forced on the caller because he cannot specify the order number as part of the keyed GET.



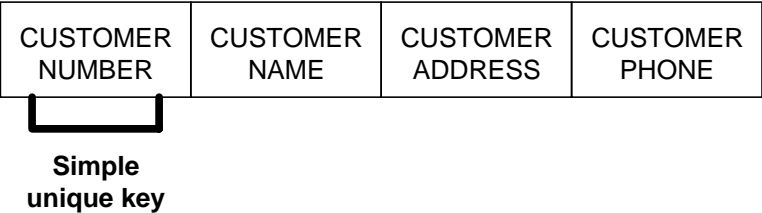
Cincom recommends that you uniquely identify a single logical row whenever possible. It is required for [INSERT](#) and [UPDATE](#) operations.

When you define a simple or compound unique key (see “[Simple unique keys](#)” on page 36 and “[Compound unique keys](#)” on page 37), the program might not supply all the values. For example, if you define the customer number and order number as a compound unique key, the program can retrieve the row using zero, one, or two key values. In this way, the program can implement a generic read by specifying zero or more key values and less than the total number of logical keys in the view. If the program specifies just customer number, the RDM would retrieve all orders for that customer.

If the logical key maps to the physical key of a file which maintains uniqueness of the physical key, the RDM will let the data manager maintain the uniqueness. If the column does not map to a unique key, the RDM tries to keep the value unique.

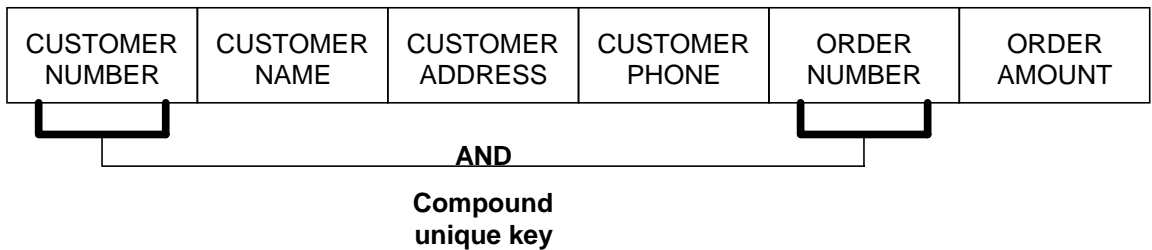
Simple unique keys

Think of a simple unique key as a selection criteria. All that occurs is an equal comparison between some column and a program-specified value. An example of a simple unique key is the customer number. No two customers for a company should have the same customer number. Therefore, the key is unique. The following figure shows columns from the customer file, pointing out customer number as a unique key.



Compound unique keys

A compound (concatenated) unique key consists of more than one column (see the following figure). Assume your view has a customer number and an order number, both defined as logical keys. At the program level, the user can code a **GET** view using customer number value and an order number value. The RDM tries to locate the row based on both values.



It is as if the program had specified **GET** customer order where customer number equals a certain value and order number equals a certain value. Physical navigation depends on how the fields are defined on the database and in the **ACCESS** statement on the Directory. For example, if customer number and order number are defined as a compound physical key, then the RDM takes the two key values, concatenates them, and does a random read on that compound value.

Another example is a customer having more than one order where you want to keep track of a particular relationship. The customer number would reside in one file and the order number in another file. In this case, the RDM directly reads in the customer number according to the program-supplied value. It then sweeps the customer order file until it finds the supplied order value. To the user program, there is no difference; the record is being retrieved as identified by the compound unique key.

The key value of a compound unique key is the combination or concatenation of the logical key values. The RDM tries to keep this combination unique. Using the customer-order example, several records for a given customer number may exist. There also may be several rows for a given order number, but only a single row for the customer number/order number combination.

Nonunique keys

Nonunique keys differ from unique keys in that the RDM does not maintain the key value as unique. The column is still required, and the user may specify a value for the column to select rows, but the same key value may return multiple rows. Do not confuse this with a generic search, which also may return several rows for a given key value.

Simple nonunique keys

Another type of key is the simple nonunique key. If you can have more than one row with the same logical key, then it is an unnormalized view and has nonunique keys. An example is a customer file in which you keep a list of notes or comments about each customer. You do not date the comments and you do not want to supply another key; but for each customer, you want to retrieve a list of comments that may have been recorded. This is a nonunique, unnormalized view because there would be multiple records with the same customer number. In this case, you could define the customer number as a nonunique key. When the program does its first **GET** using some customer number, the RDM retrieves the first comment for that customer. A subsequent GET retrieves the second comment; the third GET, the third comment, and so on. When the RDM reaches the last note for that customer, it reaches a boundary condition, and returns a “not found” status to the program.

Compound nonunique keys

A compound nonunique key is an extension of the simple nonunique key in that you have more than one column defined as the key and at least one of the keys is nonunique. All the nonunique keys together still do not completely describe the row occurrence as unique. You can still have more than one row with that same compound nonunique key.

A nonunique key, in combination with other keys of either type (unique or nonunique), forms a nonunique key.

Constant keys

You can supply a logical key with a fixed value, called a constant, by entering the keyword `CONST` or `UNIQUE CONST` in the column definition and then assigning a literal value to the column. The RDM uses this value as though the program had supplied the value as a key. You can use a `UNIQUE CONST` column to prevent duplication on inserts for the constant value specified. Constant columns must pass data validity checking if checking is specified, and may not be null. A constant key column is always a required column in the view.

You can use a constant key for value-based security. For example, you want to define a view that retrieves only the customers from Tennessee. You could supply a constant of `TENN` (or whatever your application required) to the state column. Then, the program can retrieve and update only Tennessee customers.

When you designate a column as a constant key, the RDM does not return the column value in the row. For example, the user of the view would never see the state value `TENN`.

Required columns

A required column must have a value that is valid and not null. It must have a value whether or not you want to retrieve the value in a given case. Every key column (every column with any of the qualifiers `KEY`, `NONUNIQUE KEY`, `CONST`, or `UNIQUE CONST`) is a required column. You can specify a nonkey column as a required column with the qualifier `REQ`.

Your programmers must know which columns have been defined as required columns since it can affect processing. If the programmer does not include a required column in the user view, the RDM still requires the column, even though it is not returned to the user view.

Using domains, null values, and default values for physical fields

Every column in a view corresponds to a physical field defined on the Directory. The definition of a physical field on the Directory specifies the field's characteristics such as length, format, validation option, default value, and null value. The RDM uses these characteristics from the Directory when processing RDML requests. Especially important in processing views are the validation option, default value, and null values because they are used to validate the data and maintain data integrity.

You can use the DBAID command **COLUMN-DEFN** to display the default value, null value, and validation criteria for each column in a view. You can also use this command to report on the physical characteristics of the column, such as length, edit masks, format, and ordering. See [“Managing views with the DBAID commands”](#) on page 133 for more information about DBAID commands.

The following sections discuss how the RDM utilizes the information on the Directory about a physical field.

Validation options

The Directory contains the validation options and defines the options the RDM should use to validate a column value when mapping a column to a physical field. The available options are:

Option		Meaning
R	Range Checking	Specifies that the RDM should verify that the column value is within a minimum and maximum range specified on the Directory.
T	Table Checking	Specifies that the RDM should verify that the column value is an entry on a validation table stored on the Directory.
E	Exit	Specifies that the RDM should utilize the specified exit to verify the value in a column.
(Blank)		No validation.

You can specify only one validation option for a particular field; options are mutually exclusive. Refer to the [SUPRA Server PDM Directory Online User's Guide \(OS/390 & VSE\)](#), P26-1260, or the [SUPRA Server PDM Directory Batch User's Guide \(OS/390 & VSE\)](#), P26-1261, for instructions for specifying validation options with Directory Maintenance.

The RDM performs validation checking before each **INSERT** or **UPDATE** whenever a column in a view corresponds to a physical field with validation. When performing retrievals on base views, the RDM checks the Retrieval Validation Flag for each physical field. The Retrieval Validation Flag is set to Y if the field should be validated on retrievals. If a column maps to a physical field and the Retrieval Validation Flag is set to Y, the RDM validates the data.

Range checking

If you specify range checking, the RDM verifies that a value in a column is within a specified range. You can specify the minimum value and the maximum value that the RDM uses to validate. The maximum length of a range value is 32 bytes. This is normally sufficient for data types other than character. For character columns that have lengths greater than 32 bytes, the range value is padded to the right with blanks during the comparison.

Table checking

If you specify table checking, the RDM verifies that a value in a column is contained within a table of values stored on the Directory. You build a table of values on the Directory and specify the name of the table for the RDM to use. The DBA must create each validation table on the Directory. Each entry in the table can be a maximum of 72 bytes long. You may use hex notation if you wish.

For example, you have ten suppliers. Whenever you place an order, the RDM verifies that the supplier you specify is one of the ten you are authorized to use. If the supplier is in the table, your order is processed.

Exits

If you specify exit validation, the RDM calls the user-written exit program whose name you supply for that field. You design and write the exit to perform whatever validation checking you need. The exit must pass a return code back to the RDM indicating whether the column's value is valid. See [“Customizing the RDM processing with user exits”](#) on page 237 for information about using validation exits.

Default validation

If a field's type is defined as packed decimal or zoned decimal, the RDM automatically verifies that the value for the field is a valid number. This check is made after the check for nulls but before doing user-specified validation, if any.

Join compatibility

The RDM ensures that any columns used in a join are from the same domain unless you explicitly override this checking. For example, you cannot join a column from a domain of numbers with a column from a domain of alphanumeric characters. The following ACCESS statement is incorrect because CUSTOMER-NO is from the CUSTOMER-NUMBER domain while CUSTOMER-NAME is from the NAME domain.

```
ACCESS E$CU WHERE CUSTOMER-NO = CUSTOMER-NAME
```

The next example uses the extra equal sign to indicate that the RDM should not perform normal domain checking. This ACCESS statement is permissible as long as CUSTOMER-NAME and CUSTOMER-NO are the same length.

```
ACCESS E$CU WHERE CUSTOMER-NAME = = CUSTOMER-NO
```

If one or both columns in a join do not have a domain, the RDM only verifies that the length of both fields is the same.

Redundant columns must be from the same domain (if all redundant columns have domains specified) unless you override this checking. You override the normal domain checking by using the optional equal sign when making columns redundant. The following example overrides the restriction that REGION-NO and BRANCH-NO must have the same domain:

```
REQ REGION-NO = = REGION-NO = BRANCH-NO
```

GET processing

When performing **GETS**, the RDM validates each column in the base view if the retrieval validation flag is set to Y. The RDM verifies the value either by checking the validation table or the range specified, or by utilizing the validation exit. The RDM returns an invalid column status indicator (ASI) for each column that fails to meet the validation criteria. Required columns must have values that are valid and not null, or the RDM does not return a row.

INSERT processing

Before processing an **INSERT** command, the RDM validates each user view column's value. The RDM returns an invalid ASI for each column that fails to meet the validation criteria. If all column values are valid, then the RDM INSERT proceeds.

UPDATE processing

Before processing an **UPDATE** command, the RDM validates each user view column's value. The RDM returns an invalid ASI for each column that fails to meet the validation criteria. If all values are valid, then the RDM UPDATE proceeds.

Null values

The Directory allows you to define, for each field, whether the field can be null and what value the field should contain to represent a null value. A null value means that a column is empty and its value has no meaning. Typically, blanks are used to represent null values. However, you can define on the Directory what value is to represent a null value for a field; the value can be blanks, or zero, or any value you specify.

GET processing

When the RDM processes a **GET** request, each column that is equal to a null value has an ASI of missing (-), and is set to zero for numeric type data and to blanks for all other data types. Required columns must not be null.

INSERT processing

When the RDM processes an **INSERT** command, all columns with a null ASI (an ASI of N) are set to their corresponding null value.

The application program can insert a null value into a column by setting the ASI to N or by supplying the null value in the column. The DBAID user can insert a null value by inserting the keyword NULL into the column or by supplying the null value.

The RDM rejects any insert that supplies a null value for a required column. The RDM allows a null value for a foreign key if the foreign key is not a required column.

UPDATE processing

When the RDM processes an **UPDATE** command, all columns with a null ASI (an ASI of N) are set to their corresponding null value.

The application program can update a null value into a column by setting the ASI to N or by supplying the null value in the column. The DBAID user can insert a null value by inserting the keyword NULL into the column or by supplying the null value.

The RDM rejects any update that attempts to change the value of a required column to a null value. The RDM allows a null value for a foreign key if the foreign key is not a required column.



If the user supplies the null value in a column, the application program is dependent on the null value.

DELETE processing

When a view deletes a primary key, the base view definition can allow for the foreign keys to be either cascade deleted or nullified, or to restrict the delete. You specify that the foreign keys should be deleted by specifying ALLOW DELETE on the access to the file containing the foreign key. Alternatively, you specify the foreign key to restrict the delete by not adding an ALLOW on the access to the file containing the foreign key. You specify that the foreign keys should be nullified by specifying ALLOW UPDATE on the access to the file containing the foreign key. This is useful, for example, if you want to delete a region but not all of the branches. Each branch's region number is set to a null value until the branches can be reassigned to a new region. See "[Maintaining referential integrity](#)" on page 87 for more information.

Default values

The RDM uses the default value for a physical field when no column in the user view maps to that physical field, either because the user view subset does not include the mapping column or because the view does not contain the mapping column. A default value for each field can be 1–32 alphanumeric characters; however, blanks are commonly used as the default. Default values are defined on the Directory. For fields larger than 32 bytes, the default value is padded on the right with blanks.

Example 1. An RDM application program inserts a new branch using the user view BRAN-USER, which is derived from the base view BRAN. When the program inserts a new branch, it does not supply a value for the BRANCH-STATE field. In this case, the RDM uses the default value specified for the BRANCH-STATE field:

BRAN base view	BRAN-USER user view
BRANCH-NO	BRANCH-NO
BRANCH-NAME	BRANCH-NAME
BRANCH-ADDR	BRANCH-ADDR
BRANCH-CITY	BRANCH-CITY
BRANCH-STATE	BRANCH-REGION
BRANCH-REGION	

The program supplies the following row:

1241	DUNCAN	124	NORTH	'B'	ST	ORANGE	777
------	--------	-----	-------	-----	----	--------	-----

The physical field that maps to BRANCH-STATE has a default value of CA. The RDM uses the BRANCH-STATE default value, CA, and inserts the following row into the database:

1241	DUNCAN	124	NORTH	'B'	ST	ORANGE	CA	777
------	--------	-----	-------	-----	----	--------	----	-----

Example 2. Another example is when a view does not contain a column that maps to a field in the physical file. In this example, the BRAN view does not include a column that maps to the BRANCH-STATE physical field. When the program inserts a new branch, it cannot supply a value for the BRANCH-STATE field. In this case, the RDM uses the default value specified for the BRANCH-STATE field:

BRANCH file	E\$BR	BRAN view
BRANCH-NO		BRANCH-NO
BRANCH-NAME		BRANCH-NAME
BRANCH-ADDR		BRANCH-ADDR
BRANCH-CITY		BRANCH-CITY
BRANCH-STATE		BRANCH-REGION
BRANCH-REGION		

The program supplies the following row:

1241 DUNCAN 124 NORTH 'B' ST ORANGE 777

The physical field that maps to BRANCH-STATE has a default value of CA. The RDM uses the BRANCH-STATE default value, CA, and inserts the following record into the database:

1241 DUNCAN 124 NORTH 'B' ST ORANGE CA 777

3

Modifying user data

This chapter describes how to change the contents of your database and how to control accesses at the file or view level. It also describes the status indicators that the RDM returns to the DBAID user or the application program when accessing and modifying your database.

Changing the database contents

It is possible to change the contents of the database in the following ways. You can:

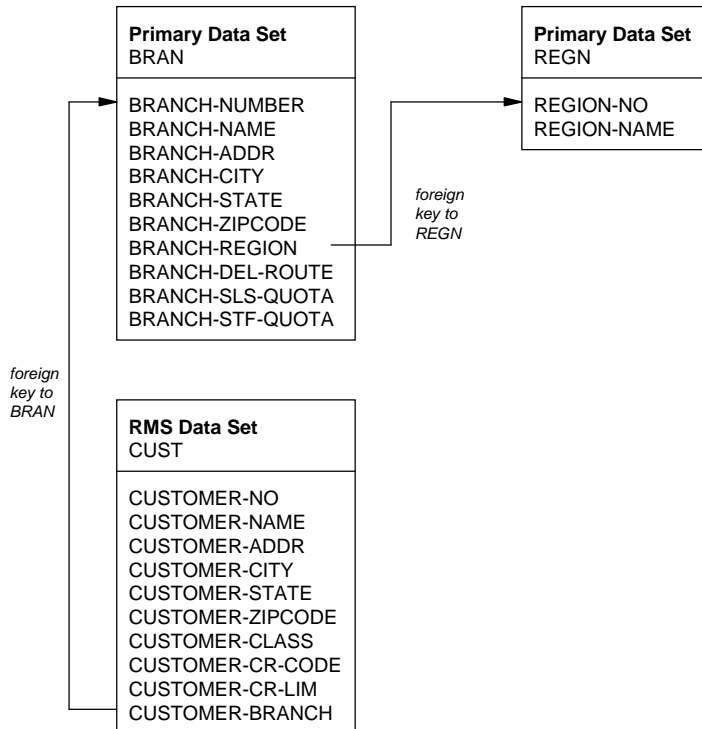
- ◆ Insert new information.
- ◆ Change existing information to new values.
- ◆ Delete information.

The ALLOW phrase of the ACCESS statement controls maintenance actions on a physical file or on a view. It specifies what maintenance is to be allowed for each file and the logical actions to be performed on a view. Therefore, you control security on a file or view level. You can use the options in the following table with the ALLOW clause (see “[Access definition](#)” on page 72 for more information):

Option	Action
INSERT	Allows insertions to the database.
UPDATE	Allows updates or replacements to the database.
DELETE	Allows deletions from the database.
SHARED	Allows column values to be shared between views (not available for use in derived views).
ALL	Allows all forms of database modification.

The following sections discuss INSERT, UPDATE, and DELETE. The examples are based on the sample database shown in the following figure.

The following figure also shows that BRANCH-NO, the primary key in the BRANCH relation, has a foreign key, CUSTOMER-BRANCH, in the CUSTOMER relation. REGION-NO, the primary key in the REGION relation, has a foreign key, BRANCH-REGION, in the BRANCH relation.



The base view for the subsequent examples is as follows:

```

> 0100 DEFINE BRANCH-VIEW
> 0200 KEY    BRANCH-NO
> 0300        BRANCH-NAME
> 0400        BRANCH-ADDR
> 0500        BRANCH-CITY
> 0600        BRANCH-STATE
> 0700        BRANCH-ZIPCODE
> 0800 REQ    BRANCH-REGION = BRANCH-REGION = REGION-NO
> 0900 ACCESS E$BR WHERE BRANCH-NO = BRANCH-NO
> 1000        ALLOW ALL
> 1100 ACCESS E$RG ONCE WHERE REGION-NO = BRANCH-REGION
> 1200 ACCESS E$CU WHERE CUSTOMER-BRANCH = BRANCH-NO
  
```

Inserting information to the database

For an insert to a relation, you can code your view to have the RDM:

- ◆ Reject the insert if the foreign key does not exist as a primary key.
- ◆ Use the foreign key to automatically insert it as a primary key in the target relation.

To insert a branch to the example database, you need a region for that branch because the BRANCH relation has a foreign key (BRANCH-REGION) that is the primary key (REGION-NO) in the REGION relation. You can have the RDM reject any insert of a BRANCH if the REGION does not exist. You can also have the RDM automatically insert a region when you insert a branch. In the example view above, the ACCESS statement

```
> 1100 ACCESS E$RG ONCE WHERE REGION-NO = BRANCH-REGION
```

provides the integrity that will not allow insert if BRANCH-REGION is not an existing primary key in the REGION relation. See “[Maintaining referential integrity](#)” on page 87 for more information about referential integrity.

The illustration under “[Changing the database contents](#)” on page 49 shows how the foreign key in the BRANCH relation references the REGION relation.

You control insertions by defining ALLOW INSERT for the appropriate files or views in your ACCESS statement. If you have not allowed for insertions on a file or view, the RDM cannot insert any rows into that file or view.

The RDM checks the validity of all columns before an insert. All required fields must be valid and not null for an insert to succeed. You can insert null only if nulls are allowed.

Updating information on the database

For an update to a relation, you can code your view to have the RDM:

- ◆ Reject the update if the foreign key does not exist as a primary key.
- ◆ Use the foreign key to automatically insert it as a primary key in the target relation.

To allow updates, define `ALLOW UPDATE` in your `ACCESS` statement for the file or view you want to update. The program can then read (**GET**) a row, change a column and issue an `UPDATE` command. The RDM updates the physical record. The RDM modifies only the physical records that have changed.

For example, assume you have a view consisting of three columns from three different files. The program gets the row, changes the value of only one of the columns, and issues the `UPDATE` command. The RDM does not do three writes to the three files; the RDM only issues one write to the affected file.

The RDM checks the validity of all columns before an update. All required fields must be present and not null for an update to succeed. You can only update a column to null if nulls are allowed.

Deleting information from the database

The RDM has three types of delete integrity:

- ◆ Restrict delete
- ◆ Cascade delete
- ◆ Nullify delete

An example of a restrict delete is trying to delete a **BRANCH** that has **CUSTOMERS** referencing it. If, on the **ACCESS** statement of the **BRANCH-VIEW** example above, you code

```
> 1200 ACCESS E$CU WHERE CUSTOMER-BRANCH = BRANCH-NO
```

then the RDM will check the **CUSTOMER** file for customers related to the branch number you want to delete. If customers exist for the branch, you cannot delete the **BRANCH**.

In a cascade delete, you would tell the RDM to delete any customers linked to the branch you are deleting. In that case, you must code **ALLOW DELETE** on the **BRANCH** and **CUSTOMER** relations, and no customer data may be in the view.

For a nullify delete, you code **ALLOW DELETE** on the **BRANCH** relation and **ALLOW UPDATE** on the **CUSTOMER** relation. Then for all customers in the branch, the RDM will set the foreign key to **BRANCH** to its null value.

You control deletions by defining **ALLOW DELETE** for the appropriate files or views in your **ACCESS** description. If you have not allowed for deletions on a file or view, the program cannot delete any rows from that file or view.

Allowing shared column values

You can allow column values to be shared between views by specifying **SHARED** on the **ALLOW** phrase of the **ACCESS** statement in base views. You cannot use **SHARED** in derived views. Using **SHARED** allows for:

- ◆ More efficient processing because automatic column value checking is bypassed when not needed.
- ◆ Modification of the same column in multiple views by the same task or other tasks. For example:

```
> DEFINE VIEW1
> 0100 KEY      BRANCH-NO
> 0200 KEY      BRANCH-REGION
> 0300          REGION-NAME
> 0400 ACCESS E$BR WHERE BRANCH-NO = BRANCH-NO
> 0500 ACCESS E$RG WHERE REGION-NO = BRANCH REGION
> 0600          ALLOW SHARED UPDATE
```

The use of **SHARED** tells the RDM that the column values from a view may be shared between views and may change between a **GET** and a later **UPDATE** or **DELETE**. When the **SHARED** phrase is present, the RDM does not check to see whether column values have changed. If **SHARED** is not on the **ALLOW** phrase for an **ACCESS** statement, then the RDM performs a check on each column in the view. This ensures that column values have not changed. The RDM does not check read-only columns that do not participate in the **UPDATE** or **DELETE**.

In the preceding example, the only maintenance function that can be performed is **UPDATE**, and the only column that can be altered is **REGION-NAME**. Because **SHARED** is part of the **ALLOW** phrase, **REGION-NAME** is automatically altered. Since **SHARED** is part of the **ALLOW** phrase, automatic hold and replace will not produce an error even if another view changes the value of the column. If the **ACCESS** statement is changed to:

```
> 0500 ACCESS E$RG WHERE REGION-NO = BRANCH-REGION ALLOW UPDATE
```

and any other view changes the column, an **UPDATE** or **DELETE** will fail. If such a failure occurs, you will receive the following message:

```
FSI: D  VSI: C  MSG: COLUMN VALUE CHANGED BY ANOTHER VIEW
```

The RDM returns the function status indicator (FSI) value **D**, indicating a data error, and the validity status indicator (VSI) value **C**, indicating that column value(s) have been changed with another view. The RDM returns the column status indicator (ASI) value **C** for each changed column. In this case, the RDM returns **C** for the **REGION-NAME** column. See “**Status indicators**” on page 58 for information about status indicators.

Retrieving data with the RDM

The view definition for a particular view defines the characteristics of the columns in the view and how to access the views or files. (See “[Maintaining referential integrity](#)” on page 87 for information on how to specify your view definition.) To properly define the access technique for each file, you need to understand how physical navigation of the database occurs.

Navigation is the act of moving from one record to another record. The record could be in the same file, or it could be in a different file. Therefore, you must understand the different relationships between records because that affects how navigation occurs from one record to the next and impacts the overall efficiency of the RDM.

You can set up your views so that the RDM uses one of the following types of database navigation methods:

- ◆ **Penetration.** Efficient access method using a direct, keyed read to the database.
- ◆ **Sweep.** Less efficient access method which involves taking a positional step forward or backward in the database.
- ◆ **Index.** Efficient access method using an index. This section describes the three types of navigation methods in further detail and discusses how positional relationships impact the access method.

The simplest relationship is the one-to-one keyed relationship. Some value from a row is used as a key to identify and access the next row. An example would be to read a branch record, select the BRANCH-REGION field, and use that value as a direct read to a REGION file. The key can be constructed from many different columns in order to get to the next row. For example, you could read FILE-A and get a piece of data, read FILE-B and get a piece of data, and put the two pieces of data together to be the compound key for FILE-C. If you are going from one or many files to another and are using a key to do your retrieval, it implies a one-to-one keyed relationship from the source to the destination.

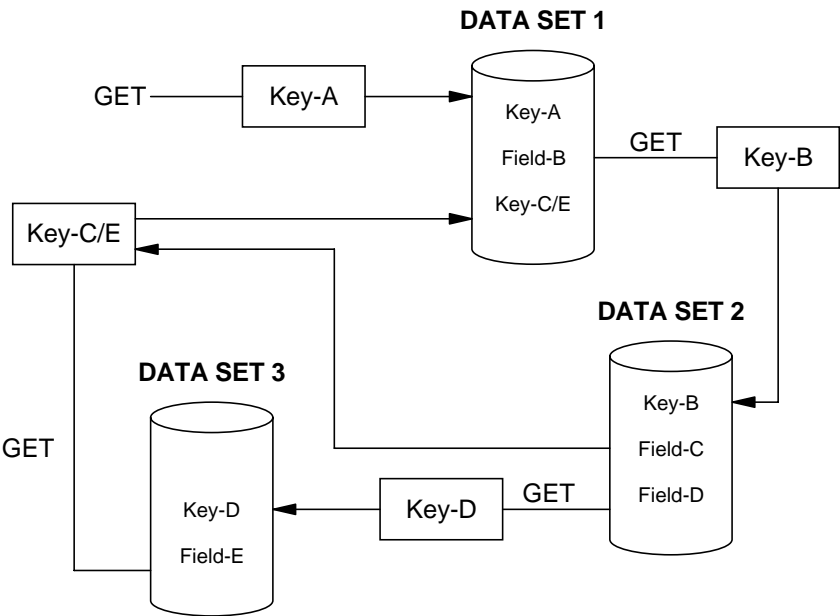
The next kind of relationship is positional relationship. This relationship implies placement, or location, as opposed to being based on some key value. For example, in a sequential file, getting the next record is a one-to-one positional relationship because you do not use a key to get it; you use the position of the first record to get to the next.

Database penetration

Database penetration is associated with the one-to-one keyed relationship. You can have the RDM penetrate (or access) the database based upon a key value, beginning at one point (or physical record) and extending outwards using the step-by-step (row-by-row) navigational method. Penetration then is accessing the database without any context of where the position was before you entered. An example of database penetration is retrieving a branch view based upon a particular branch number as the key. This retrieval does not rely on anything you have done with the database prior to that point.

Database penetration occurs when an application or user performs a keyed **GET** or a GET to establish position within the database. After the RDM establishes position based on the logical keys, the user may perform a positional GET without keys or another “penetrating” GET with keys. A GET first or last row is guaranteed to penetrate the database, while GET NEXT or PRIOR is positional.

Penetration involves the base file or view that is your starting point. This is the first record you access. From that record, you travel outward in one or more directions. Each time you take a step, you use that information to take additional steps. This would resemble a tree structure. However, it can come back together to the starting point (as shown in the following figure) or any other place along the path.



Database sweep

A database sweep is taking a positional step either forward or backward. The database sweep only occurs on a one-to-many positional relationship. Sweeping can occur when you have already penetrated the database and are positioned at some set of records. For example, if you have a view consisting of regions and branches, you first penetrate the database based on the region number. When you ask to look at the next and subsequent branches, you are performing a database sweep based on a positional relationship.

A sweep always uses an incremental movement, either forward or backward. You can sweep a file without having first penetrated the database by starting with the very first or the very last record. This is called implied penetration.

Indexing

Using an index is an efficient way to access the database. An index allows the RDM to penetrate the database based on an index value and then move positionally through the database, retrieving records in index value order.

Navigational constraints and boundaries

The RDM allows you to identify certain points along the navigation path which you must reach for the navigation to be valid. Once you reach those points, you can also identify certain values that must exist by specifying certain fields as required. The navigation is unsuccessful if the RDM does not find the fields.

Logical key columns are always required columns. Therefore, if you are trying to access a file based on a key value and the RDM does not find that value, navigation stops. If you are attempting a database penetration and the RDM does not find any required columns, the RDM returns a “not found” status to the program. However, if you are attempting a sweep through the database, the RDM skips the rows that do not meet the constraints and does not return them as part of that view. An example is the region-branch view which only returns records for regions that have at least one branch.

Certain boundary conditions exist when you do incremental movement. When you are incrementally sweeping a keyed file, the end of the file is a boundary. For example, if you are sweeping through a PDM related file chain (based on a primary file key), the end of the chain is a boundary. If, however, you sweep the related file not based on a specific primary file key, the RDM will get the next primary file record and navigate through its associated related file chain.

Status indicators

The RDM returns status indicators to the application program or to the DBAID user to indicate Relational Data Manipulation Language (RDML) processing results. The indicators are the same, regardless of whether the view is a base or derived view. Base views pass the indicators up to the derived view.

The types of status indicators are as follows:

- ◆ **FSI (function status indicator).** Returned after any RDML function call and indicates the success or failure of the function.
- ◆ **ASI (column status indicator).** Returned after a **DELETE**, **INSERT**, **GET**, or **UPDATE** RDML function call and indicates the status of each column in the row.
- ◆ **VSI (validity status indicator).** Returned after a **DELETE**, **INSERT**, **GET**, or **UPDATE** RDML function call and indicates the most severe column status within the row.

Function status indicators

A function status indicator (FSI) reflects the success or failure of the RDML function executed. The RDML processor returns the FSI to the program in an area generated as part of the programmer-supplied TIS-CONTROL statement. The following shows a COBOL example of this generation (the asterisk indicates the statement the programmer specifies; the RDML compiler generates all other statements):

```
*01  INCLUDE TIS-CONTROL.
01  TIS-CONTROL.
10  TIS-OBJECT-NAME      PIC X(30).
10  TIS-OPERATION.
    15  TIS-ID           PIC X(2).
    15  TIS-OPCODE       PIC X.
    15  TIS-POSITION     PIC X.
    15  TIS-MODE         PIC X.
    15  TIS-KEYS         PIC X.
10  TIS-FSI             PIC X.
10  TIS-VSI             PIC X.
10  FILLER              PIC X(2).
10  TIS-MESSAGE         PIC X(40).
10  TIS-PASSWORD        PIC X(8).
10  TIS-OPTIONS         PIC X(4).
10  TIS-CONTEXT         PIC X(4).
10  TIS-LVCONTEXT       PIC X(4).
```

The following table lists the meanings of the FSI values:

FSI	Meaning
*	Successful completion. The RDML function has completed successfully.
D	Data error. The row contains invalid data.
F	Failure. The RDML function has failed. Usually caused by a physical database problem returned to the RDM.
N	Not found. The RDML processor could not find an occurrence of the requested row.
S	Security check. The attempted RDML function violated a security constraint.
U	Unavailable resource. The resource required to complete this function was not available, for example, file not open.
X	Reset recommended. While processing, RDML functions modified the database before the RDM detected the error condition. Issue a RESET to restore the database. This code overrides D, F, S, or U indicators.

If the RDML processor returns an FSI value of D, check the ASIs to see which columns contain invalid data. A message associated with the FSI is accessible in the TIS-MESSAGE area of TIS-CONTROL for all returned indicators (see the preceding example).

Column status indicators

The column status indicators are called ASIs. (The A stands for attribute, which is the same thing as a column.) Each ASI reflects the status of each column defined in your view. ASIs have a one-to-one mapping to each column and are placed immediately following the last column in the view, for example:

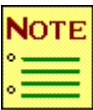
COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4	ASI ₁	ASI ₂	ASI ₃	ASI ₄
----------	----------	----------	----------	------------------	------------------	------------------	------------------

You can access the ASIs through COBOL-assigned names generated by the RDML compiler. The application programmer codes the program, specifying an INCLUDE statement for the view required. The RDML compiler generates a statement for each column included in the view. The RDML compiler also generates a statement for each required ASI column by preceding each column name with the four characters ASI-. The following shows an example of this generation (the asterisk indicates the statement the programmer specifies; the RDML compiler generates all other statements):

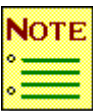
```
*01  INCLUDE CUST-CONTACT.
    01  RDM-CUST-CONTACT.
        10  CUST-CONTACT.
            20  CUST-NO                                PIC S9(05).
            20  CONTACT-NAME                           PIC X(040).
            20  CONTACT-TITLE                          PIC X(040).
            20  CONTACT-PHONE                          PIC S9(10).
        10  ASI-CUST-CONTACT.
            20  ASI-CUST-NO                             PIC X.
            20  ASI-CONTACT-NAME                       PIC X.
            20  ASI-CONTACT-TITLE                      PIC X.
            20  ASI-CONTACT-PHONE                      PIC X.
```

The following table lists the meanings of the ASI values:

ASI	Meaning
+	The column exists and a newly accessed record provided its value. This ASI value is meaningful for GET processing only.
=	The column exists and a previously accessed record provided its value. This ASI value applies to GET processing only.
-	The value for this column is null. Either the physical record field contains the value defined as null, or no physical record exists to supply this column value. The RDM returns spaces or zero, depending on the column's data type, as the column's data. It does not return the actual null value. This ASI value allows you to distinguish a null column from a column that actually contains spaces. This ASI value applies only to GET requests.
C	Another view has changed the value for this column. The RDM checks for this when an UPDATE or DELETE command follows a GET command other than GET FOR UPDATE. You can override this check by specifying SHARED in the ALLOW clause of the access definition in the view
N	The application has placed an N in the ASI to set a column to its null value as part of an UPDATE or INSERT operation. The RDM never returns an ASI of N.
V	The value for this column is invalid for one of two reasons: <ol style="list-style-type: none"> 1. The value does not meet the validation criteria defined for this column. Refer to “Validation options” on page 41 for information on validation criteria. 2. The column is a foreign key, and the corresponding primary key for this value cannot be found. Refer to “Maintaining referential integrity” on page 87 for information on foreign keys and referential integrity.



After an **INSERT** or an **UPDATE**, C and V are the only meaningful ASIs. ASIs that appear after a **DELETE** function have no meaning.



The ASI values plus (+) and equal (=) do not depend upon the state of the data area that maps to the row. The occurrence in the physical database of a record determines the ASI value. When you read another row, the value of a column may not change, but since you have read a new physical database record, the ASI is +. For example, if you are reading all of the branches for a region, each time you read a branch, the ASI for the branch number column is + even though the region number did not change. These values only have meaning on **GET** RDML requests; on **UPDATE**, **INSERT** or **DELETE** requests, they are set to +. Therefore, application programs should not depend on the value of these ASIs.

Validity status indicators

Validity status indicators (VSIs) reflect the validity of a view after a RDML command causes a read of the physical database. The RDML processor returns the VSI value to the program in an area generated as part of the programmer- supplied TIS-CONTROL statement (refer to the example in “Function status indicators” on page 59). The VSI value for a function is the same as the most serious of the ASI values the RDM returns for the columns. The ASI values the RDM can return, in order of decreasing seriousness, are: C, V, -, +, =. (The RDM never returns the ASI value N.) The following table is in the same order:

VSI	Meaning
C	Another view changed a column value.
V	The RDM is returning at least one invalid ASI.
-	The RDM is returning no invalid ASIs, but is returning at least one missing (null) ASI.
+	The RDM is returning no invalid or missing ASIs, but is returning at least one new physical occurrence in the database.
=	This RDM function is returning no invalid ASIs, missing ASIs, or new physical occurrences.

The VSI enables the programmer to quickly determine if any additional processing of ASIs is needed to correct invalid data or to supply missing values.

4

Defining and using derived views

Derived views access base views or other derived views as sources of data. You must define base views for derived views to access. Base views access the physical files and specify all integrity constraints between those files. You can build base views with DBAID or Directory Maintenance. Then you can build on these base views to derive other views without respecifying the integrity constraints. You can tailor derived views for different users and impose additional security.

With the RDM reports, you can list all the base views defined for a given schema on the Directory. See [“Using the RDM reports”](#) on page 211 for information on RDM reports. This list of base views can help you design derived views to fill the needs of the users. You can use DBAID to define derived views and test them before saving them on the Directory.

You need not store base views on the Directory before creating derived views that access them. A derived view can access an opened base view whether that base view has been saved or not. Once a base view is open, though not necessarily saved, it can be accessed. Therefore, you can create both base views and derived views in a test environment, and test them before modifying the Directory.

Defining derived views

The view definition contains the following types of statements:

- ◆ Column definitions
- ◆ Access definitions

Column definitions define each column included in the view. Access definitions define how to access the views to obtain column values. You can create these definitions using the DBAID utility (see “[Signing on to DBAID and RDM](#)” on page 105 for a sample DBAID session) or through the Directory Maintenance Access Set category using the VARIABLE EDIT command. (Refer to the [SUPRA Server PDM and Directory Administration Guide \(OS/390 & VSE\)](#), P26-2250, for instructions for defining views with Directory Maintenance.) All column definitions must precede the first access definition in the view definition. While the column definitions need not be in any particular order, define the logical keys in the order that the supplying files are accessed.

Column definition

The column definitions, entered as part of the view definition, define each column to include in the derived view and each column’s characteristics. You must define the column name of each column to be included in a particular derived view.

$\left\{ \left[\begin{array}{l} \text{REQ} \\ \text{[NONUNIQUE]} \end{array} \right] \right\}$	$\left\{ \left[\begin{array}{l} \text{column} = [=] \text{source}_1 \\ \text{column} = [=] \text{source}_1 = [=] \text{source}_2 \text{ [...]} \end{array} \right] \right\}$	
$\left[\text{[UNIQUE]} \text{CONST} \right]$	$\left\{ \left[\begin{array}{l} \text{column} = [=] \text{source}_1 \\ \text{column} = [=] \text{source}_1 = [=] \text{source}_2 \text{ [...]} \end{array} \right] \right\}$	$\left. \right\} \text{constant}$

[REQ
[NONUNIQUE] KEY]

Description	<i>Optional.</i> A qualifier specifying the characteristics of a column.	
Options	REQ	This column is not part of the logical key.
	KEY	This column is part of the logical key. The logical key is unique.
	NONUNIQUE KEY	This column is part of the logical key. The logical key need not be unique.

Considerations

- ◆ A column with one of these qualifiers is a required column. See the General Considerations for the effects of required columns.
- ◆ You can specify a maximum of nine KEY and NONUNIQUE KEY columns in a view.
- ◆ If you specify KEY or NONUNIQUE KEY, you must not specify CONST or UNIQUE CONST for this column.

[UNIQUE] CONST

Description	<i>Optional.</i> A qualifier specifying the characteristics of a column that has an assigned constant value. A column with this qualifier is a required column and is part of the logical key. Indicates that this column is required in the derived view, and the value of the column must be equal to the given constant for the row to qualify.	
Options	CONST	This column is part of the logical key. The logical key need not be unique.
	UNIQUE CONST	This column is part of the logical key. The logical key is unique.

Considerations

- ◆ If you specify CONST or UNIQUE CONST, you must supply a constant value for the column in the definition. If you supply a constant value for the column, you must specify CONST or UNIQUE CONST.
- ◆ All CONST and UNIQUE CONST columns are part of the logical key.
- ◆ CONST and UNIQUE CONST columns are not returned in the row.
- ◆ If you specify CONST or UNIQUE CONST, you must not specify KEY or NONUNIQUE KEY for this column.

column = [=]

Description *Conditional.* Names the access column in the view being defined, used later in the application program.

Format 1–30 alphanumeric characters and the special characters #, -, _, and \$. The first character must be alphabetic or a special character. If the first character is a # or \$, the second character must be alphabetic.

Considerations

- ◆ This option allows you to assign a descriptive, meaningful name to the application.
- ◆ Column names need be unique only within the view.
- ◆ If you do not specify a column name, RDM uses the source column name.
- ◆ If you specify redundant source column names, you must specify a column name.
- ◆ The column and its source(s) must be from the same domain, unless you override domain checking by coding the additional equal sign.

source₁

Description *Required.* Indicates the name of the column being accessed in a source.

Format The name of an existing column in a view

[>=]source₂[...]

Description *Optional.* Specifies one or more access columns, called redundant columns, that will map to a single column in the view.

Format The name of an existing column in a view

Considerations

- ◆ This is a convenient method of mapping the same value to many columns.
- ◆ If you specify multiple source-columns, you must specify a *column-name*.
- ◆ If you designate the column as a KEY, REQ, CONST, or NONUNIQUE KEY, all of the *column-names* you specify will have the same constraint.
- ◆ RDM accesses the columns according to the order specified in the ACCESS statements (see “[Access definition](#)” on page 72), which does not have to be the same order specified on this statement.
- ◆ When using [GET](#) to retrieve a row, the values of the columns in the view will be those of the last column accessed. The only exception is if the column is a KEY, CONST, or NONUNIQUE KEY with the key value given. In this case, RDM compares each redundant column with the key value before returning the row.
- ◆ The columns must be from the same domain unless you specify an override. To override the normal domain checking, include the additional equal sign as shown:

```
REQ REGION-NO = = BRANCH-REGION = REGION-NO
```

= *constant*

Description *Conditional.* Specifies the value to be assigned as a constant for this column.

Format Specify the value as:

X'nnnnnn' Hexadecimal

nnnnnnnn Numeric (binary, packed, or zoned)

'cccc' Character

Considerations

- ◆ The length of the value depends on the length of the column you are defining.
- ◆ The constant is required when you specify CONST.
- ◆ The constant must pass validity checking if the column has associated validation.
- ◆ The constant cannot be the null pattern.

General considerations

- ◆ Any columns specified as REQ, KEY, NONUNIQUE KEY, CONST, or UNIQUE CONST are required columns for the derived view.
- ◆ If a view contains two columns and one is a key and the other a nonunique key, the view will be processed as if both were nonunique keys. A nonunique key column makes the entire compound key nonunique.
- ◆ All column definitions must precede the first access definition in the view.
- ◆ Column definition statements may be in any order.
- ◆ Required columns restrict the number of occurrences in the view.
- ◆ Depending on the command, required columns affect the operation of RDM in these ways:

Command	Effect
GET	All required columns must be valid and non-null. If not, RDM takes the NOT FOUND option on direct GETs. For sweeping GETs, RDM skips the row. If a required column is not included in a user view, the required column must still be present, but RDM does not return it to the program.
INSERT or UPDATE	All required columns must be valid and non-null. If a required column is not included in the user view, RDM returns a data error.
DELETE	No effect.

- ◆ See “Required columns” on page 39 for an example of how the REQ option affects processing.

Examples

- ◆ The column definition for this view indicates a customer-product which may have multiple product codes for each customer.

```
> 0100 KEY          CUSTOMER-NO
> 0200              CUSTOMER-NAME
> 0300 NONUNIQUE KEY PRODUCT-CODE
> 0400              PRODUCT-DESC
> 0500              PRODUCT-PRICE
```

- ◆ This example shows the use of multiple column names.

```
> 0100 BRANCH = CUSTOMER-BRANCH = BRANCH-NO
```

- With a **GET**, the value returned in BRANCH depends on which column (CUSTOMER-NO or BRANCH-NO) is accessed last. RDM does not guarantee that these two values are equal in this case.
- An **INSERT** of a value into BRANCH results in the same value being inserted into CUSTOMER-BRANCH and BRANCH-NO in the accessed views.
- With an **UPDATE**, a change in BRANCH-NO updates both CUSTOMER-BRANCH and BRANCH-NO.

```
> 0100 KEY BRANCH = CUSTOMER-BRANCH = BRANCH-NO = INVOICE-BRANCH
```

RDM treats all three columns as keys:

- With a **GET**, you will retrieve only those rows that have CUSTOMER-BRANCH, BRANCH-NO, and INVOICE-BRANCH equal to the value given for BRANCH in the USING phrase. If you do not supply a key value on the GET command, RDM does not guarantee that these values are equal.
- An **INSERT** of a value into BRANCH results in the same value being inserted into all three columns (CUSTOMER-BRANCH, BRANCH-NO, and INVOICE-BRANCH).

Access definition

The access definitions determine how to get from view to view, how to access base and derived views, and the relationships you can have between views in your access statement. Enter access definitions after the column definitions.

If you use the WHERE clause without the USING clause, RDM determines the best access strategy and uses it. If you use the USING clause, RDM performs a keyed read. If you use both the WHERE and the USING clauses, RDM performs a keyed read and applies the additional selection criteria indicated by the WHERE clause.

```
ACCESS view-name [ONCE]
  [USING (value1,value2...)]
  [WHERE column1=[=]value1 AND column2=[=]value2...]
  [GIVING column1 column2 ...]
  [ ALLOW  [ INSERT ] [ REP ]
           [ DELETE ] [ UPDATE ] ]
```

view-name

Description	Required. Identifies the base or derived view to access.
Format	Must be the name of an existing view.

ONCE

Description	Optional. Indicates that you want to retrieve only the first row.
-------------	---

[USING (value₁, value₂...)]

Description *Optional.* Indicates that a logical keyed read using specified value(s) is to be done on the view.

Format Each value is a constant or an existing column.

Considerations

- ◆ In an access definition accessing a derived view, you must use the WHERE clause, the USING clause, or both. In an access definition accessing a base view, Cincom recommends you use the WHERE clause, the USING clause, or both.
- ◆ The value(s) you specify must correspond to the logical key of the accessed view. When specifying several values, you may omit values from the right hand side of the group of values. Subdefinitions of logical keys are not allowed. For example, if you had the following base view, VIEW-A, defined:

```
VIEW-A
KEY ATTR1
KEY ATTR2
KEY ATTR3
.
.
```

The following derived view, accessing VIEW-A, is valid:

```
VIEW-B
KEY ATTR-A
KEY ATTR-B
KEY ATTR-C
ACCESS VIEW-A
USING (ATTR-A, ATTR-B)
```

However, the following view, VIEW-C, is not valid because values were omitted and they do not map to logical key columns, from left to right:

```
VIEW-C
KEY ATTR-A
KEY ATTR-B
KEY ATTR-C
ACCESS VIEW-A
USING (ATTR-A, ,ATTR-B)
```

To correct VIEW-C, exchange the USING clause for the following WHERE clause:

```
WHERE (ATTR1 = ATTR-A)
AND (ATTR3 = ATTR-B)
```

or:

```
USING (ATTR-A)
WHERE (ATTR3 = ATTR-B)
```

- ◆ If the view that you are accessing only has one logical key column, you may omit the parentheses.

[WHERE *column*₁[*=*]value₁ AND *column*₂[*=*]value₂...]

Description	<i>Optional.</i> Specifies the desired values for certain columns in this view. RDM selects only those rows with the specified values.	
Format	<i>column</i>	Must be the name of a column in the view named in the ACCESS statement.
	<i>value</i>	The name of a column in this view or a previously accessed view, a constant, or a logical key.

Considerations

- ◆ In an access definition accessing a derived view, you must use the WHERE clause, the USING clause, or both. In an access definition accessing a base view, Cincom recommends you use the WHERE clause, the USING clause, or both.
- ◆ Each column and its specified comparison value must belong to the same domain unless you override domain checking with the extra equal sign.
- ◆ Use RDM statistics to measure the performance of the derived view. Refer to the description of the DBAID command **STATS** in “Managing views with the DBAID commands” on page 133 for information on using statistics.

[GIVING *column*₁ *column*₂ ...]

Description	<i>Optional.</i> Overrides the normal data movement.
Format	The keyword GIVING followed by one or more column names as defined by the column definitions

Considerations

- ◆ The GIVING clause allows you to access a view more than once and retrieve selected columns during each access. For each view that occurs on more than one ACCESS statement and contains needed columns, you can specify which columns to fill on which access of the view.
- ◆ If you omit column names on the GIVING clause, RDM uses the view for access only.
- ◆ If you omit this clause, all columns derived from accessed columns in the accessed view that have not been supplied by some previous ACCESS statement are filled with values using this ACCESS statement.

ALLOW

INS	ERT
DEL	ETE

REP
UPDATE

Description *Optional.* Specifies what RDML actions you want to allow on the accessed view.

Format Any combination is valid, for example:

ALLOW INSERT DELETE Allows inserts and deletes but not updates.

ALLOW UPDATE Allows updates but not inserts or deletes.

Options ALL Allows all three types of data modification.

INSERT Allows row inserts on the view.
INS

DELETE Allows row deletes from the view.
DEL

UPDATE Allows row updates or replacements on the view.
UPD
REP

Considerations

- ◆ RDM always allows read access to the accessed view. If you omit the ALLOW clause, RDM allows only read access to the view.
- ◆ RDM allows modification of accessed data only if each relevant access definition at each view level allows that type of modification. For example, if the relevant access definition in the referencing derived view specifies ALLOW ALL, and the relevant access definition in the accessed base view specifies ALLOW UPDATE, RDM allows updates but not inserts or deletes on the indicated data.

Examples of derived view definitions

This section presents a sample database and shows how to define and use derived views to access the sample database.

Base relations

The example conceptual schema contains five relations: **BRANCH**, **CUSTOMER**, **PRODUCT**, **REGION**, and **STOCK**. These relations contain the columns and values shown. The examples in the rest of this chapter are built on these relations.

Relation: REGION (REGN) Type: Independent Entity

Columns	Primary key	Foreign relation
REGION-NO	Y	
REGION-NAME		

Relation: BRANCH (BRAN) Type: Dependent Entity

Columns	Primary key	Foreign relation
BRANCH-NO	Y	
BRANCH-NAME		
BRANCH-ADDR		
BRANCH-CITY		
BRANCH-STATE		
BRANCH-ZIPCODE		
BRANCH-REGION		REGION
BRANCH-DEL-ROUTE		
BRANCH-SLS-QUOTA		
BRANCH-STF-QUOTA		

Relation: STOCK (STCK) Type: Relationship

Columns	Primary key	Foreign relation
STOCK-BRANCH	Y	BRANCH
STOCK-PRODUCT	Y	PRODUCT
STOCK-QNTY		
STOCK-BIN-LOC		
STOCK-YTD-SALES		

Relation: CUSTOMER (CUST) Type: Independent Entity

Columns	Primary key	Foreign relation
CUSTOMER-NO	Y	
CUSTOMER-NAME		
CUSTOMER-ADDR		
CUSTOMER-CITY		
CUSTOMER-STATE		
CUSTOMER-ZIPCODE		
CUSTOMER-CLASS		
CUSTOMER-CR-CODE		
CUSTOMER-CR-LIM		
CUSTOMER-BRANCH		BRANCH

Relation: PRODUCT (PROD) Type: Independent Entity

Columns	Primary key	Foreign relation
PRODUCT-CODE	Y	
PRODUCT-DESC		
PRODUCT-WH-QNTY		
PRODUCT-PRICE		
PRODUCT-GROUP		

Base views

Base views that represent “base relations” describe conceptual schema information. The DBA can define base views with the DBAID utility (see “[Maintaining the RDM](#)” on page 103 and “[Managing views with the DBAID commands](#)” on page 133) or Directory Maintenance. The access sets for the base views for the example database are shown following. Using these access statements, RDM optimizes the physical navigation path.

◆ Base View: REGION

View Text:

```
> DEFINE REGION
> 0100 KEY      REGION-NO
> 0200          REGION-NAME
> 0300 ACCESS E$RG WHERE REGION-NO = REGION-NO  ALLOW ALL
> * To restrict deletions of REGIONS that contain branches, code:
> 0500 ACCESS E$BR WHERE BRANCH-REGION = REGION-NO
```

◆ Base View: BRANCH

View Text:

```
> DEFINE REGION
> 0100 KEY      BRANCH-NO
> 0200          BRANCH-NAME
> 0300 REQ      BRANCH-REGION = BRANCH-REGION = REGION-NO
> 0400          BRANCH-SLS-QUOTA
> 0500          BRANCH-STF-QUOTA
> 0600          DEL-ROUTE
> 0700          BRANCH-ADDR
> 0800          BRANCH-CITY
> 0900          BRANCH-STATE
> 1000          BRANCH-ZIPCODE
> 1100 ACCESS E$BR WHERE BRANCH-NO = BRANCH-NO    ALLOW ALL
> * To verify that BRANCH-REGION contains a valid region on
> * INSERTs and UPDATEs, code:
> 1200 ACCESS E$RG ONCE WHERE REGION-NO = BRANCH-REGION
> * To restrict deletions of branches containing customers, code:
> 1300 ACCESS E$CU WHERE CUSTOMER-BRANCH = BRANCH-NO
> * To restrict deletions of branches that have stock, code:
> 1400          ACCESS E$SK WHERE STOCK-BRANCH = BRANCH-NO
```

◆ Base View: STOCK

View Text:

```

> DEFINE STOCK
> 0100 KEY      STOCK-BRANCH  = STOCK-BRANCH  = BRANCH-NO
> 0200 KEY      STOCK-PRODUCT = STOCK-PRODUCT = PRODUCT-CODE
> 0300          STOCK-QNTY
> 0400          STOCK-BIN-LOC
> 0500          STOCK-YTD-SLS
> 0600 ACCESS E$SK WHERE STOCK-BRANCH  = STOCK-BRANCH
> 0700          AND STOCK-PRODUCT = STOCK-PRODUCT      ALLOW ALL
> * To verify that STOCK-BRANCH contains a valid branch on
> * INSERTs, code:
> 0800 ACCESS E$BR ONCE WHERE BRANCH-NO = STOCK-BRANCH
> * To verify that STOCK-PRODUCT contains a valid product code
> * on INSERTs, code:
> 0900 ACCESS E$PD ONCE WHERE PRODUCT-CODE = STOCK-PRODUCT

```

◆ Base View: CUSTOMER

View Text:

```

> DEFINE CUSTOMER
> 0100 KEY      CUSTOMER-NO
> 0200          CUSTOMER-CR-CODE
> 0300          CUSTOMER-CR-LIM
> 0400 REQ      CUSTOMER-BRANCH = CUSTOMER-BRANCH = BRANCH-NO
> 0500          CUSTOMER-ADDR
> 0600          CUSTOMER-NAME
> 0700          CUSTOMER-CLASS
> 0800 ACCESS E$CU WHERE CUSTOMER-NO = CUSTOMER-NO
> 0900          ALLOW ALL
> * To verify that CUSTOMER-BRANCH contains a valid branch on
> * INSERTs and UPDATEs, code:
> 1000 ACCESS E$BR ONCE WHERE BRANCH-NO = CUSTOMER-BRANCH

```

◆ Base View: PRODUCT

View Text:

```
> DEFINE PRODUCT
> 0100 KEY      PRODUCT-CODE
> 0200          PRODUCT-WH-QNTY
> 0300          PRODUCT-PRICE
> 0400          PRODUCT-DESC
> 0500 ACCESS E$PD WHERE PRODUCT-CODE = PRODUCT-CODE
> 0600          ALLOW ALL
> * To restrict deletions of products that contain stock, code:
> 0700 ACCESS E$SK WHERE STOCK-PRODUCT = PRODUCT-CODE
```

Derived views

You can use the base views from the previous illustration to derive other views. The derived views may contain columns from the base views and columns from several other views, and may have different update options. The following three examples show increasing complexity when defining derived view access definitions.

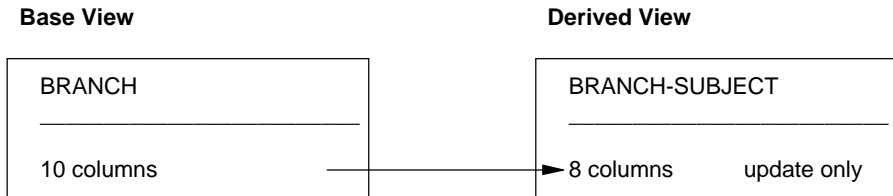
Example 1. This example creates a new view which is a subset of the BRANCH base view, and excludes the BRANCH-SLS-QUOTA and BRANCH-STF-QUOTA columns.

Derived View: BRANCH-SUBSET

View Text:

```
> DEFINE BRANCH-SUBSET
> 0100 KEY      BRANCH-NO
> 0200          BRANCH-NAME
> 0300 REQ      BRANCH-REGION
> 0400          BRANCH-DEL-ROUTE
> 0500          BRANCH-ADDR
> 0600          BRANCH-CITY
> 0700          BRANCH-STATE
> 0800          BRANCH-ZIPCODE
> 1000 ACCESS BRANCH WHERE BRANCH-NO = BRANCH-NO
> 1100          ALLOW UPDATE
```


The following figure shows the base view and the number of columns it contains, and the derived view including the number of columns used from the base view. It also shows the update options specified for the derived view.



This view can be used by users restricted from seeing the two quota fields. When defining this view, you do not have to enter all of the access statements that provide the integrity constraints, nor must you rewrite this view if the physical file for the BRANCH relation were broken apart or put into another file with a different name.

The KEY indicator is required to indicate the column RDM is to use as the logical key for this view. BRANCH-REGION does not have to be REQ in this view, but the base view only returns and accepts non-null, valid data for the column. Also, if you use REQ, you can validate the required column in the derived view, possibly avoiding the need for restoring the database.

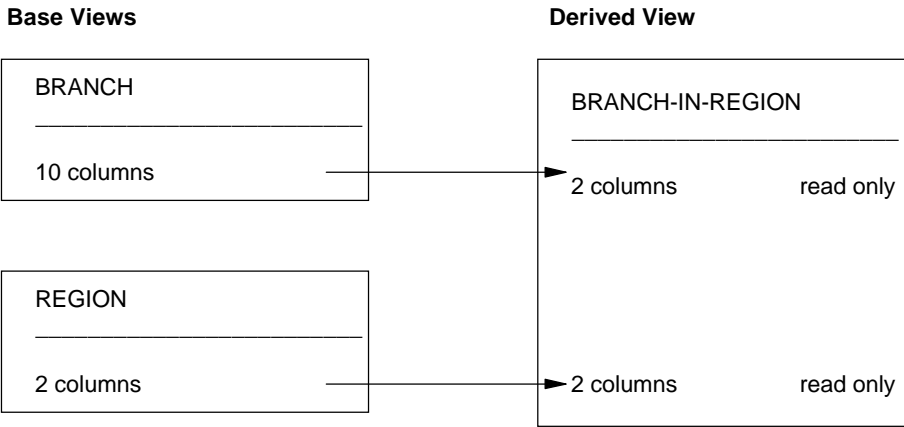
Example 2. The process becomes more complex when using several base views to build a single derived view. This example combines the REGION and BRANCH views into a composite, listing the branches within a region.

Derived View: BRANCHES-IN-REGION

View Text:

```
> DEFINE BRANCHES-IN-REGION
> 0100 KEY      REGION-NO
> 0200          REGION-NAME
> 0300 KEY      BRANCH-NO
> 0400          BRANCH-NAME
> 0500 ACCESS REGION WHERE REGION-NO = REGION-NO
> 0600 ACCESS BRANCH WHERE BRANCH-REGION = REGION-NO
```

The following figure shows the base views and the number of columns in each, and the derived view including the number of columns used from each base view. It also shows the updating options allowed for the derived view.



In addition to using two views to create a third view, this example changes the updating options for the REGION and BRANCH views. Even though REGION and BRANCH are updateable, the BRANCHES-IN-REGION view is read only. When accessing a base view with a derived view, you can make view update capability more restrictive, but not less restrictive. For example, if the BRANCH base view did not have an ALLOW statement in its access set, it would not allow updates regardless of the ALLOW statements coded on the derived views using it.

Example 3. This example lists all the products in stock in a region. The derived view accesses four base views for each row, and allows the user to perform different updating options on several of the base views. The following figure shows the base views and the number of columns in each, the derived view including the number of columns used from each base view, and the updating options specified.

Derived View: PRODUCTS-IN-REGION

View Text:

```
> DEFINE PRODUCTS-IN-REGION
> 0100 KEY      REGION-NO
> 0200          REGION-NAME
> 0300 KEY      BRANCH-NO
> 0400          BRANCH-NAME
> 0500 KEY      STOCK-PRODUCT
> 0600          PRODUCT-DESC
> 0700 ACCESS REGION WHERE REGION-NO      = REGION-NO
> 0800          ALLOW UPDATE DELETE
> 0900 ACCESS BRANCH WHERE BRANCH-REGION = REGION-NO
> 1000          ALLOW ALL
> 1100 ACCESS STOCK WHERE STOCK-BRANCH    = BRANCH-NO AND
> 1200                                STOCK-PRODUCT = STOCK-PRODUCT
> 1300          ALLOW ALL
> 1400 ACCESS PRODUCT WHERE PRODUCT-CODE = STOCK-PRODUCT
```

Base Views

REGION
2 columns

BRANCH
10 columns

STOCK
5 columns

PRODUCT
4 columns

Derived View

BRANCH-IN-REGION	
→ 2 columns	update and delete options
→ 2 columns	all update options
→ 1 column	all update options
→ 1 column	read only

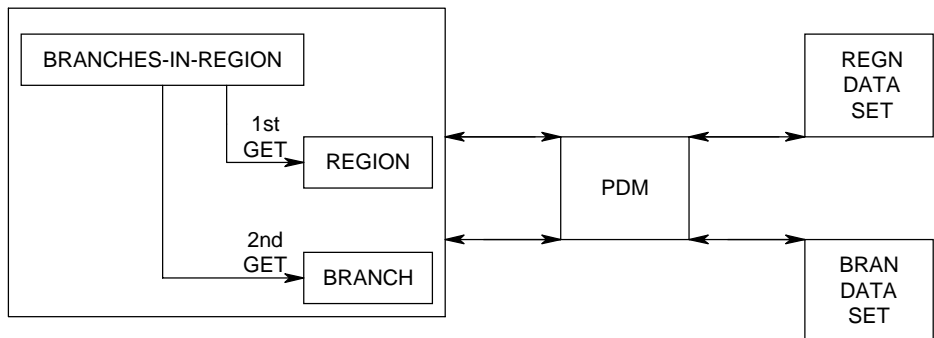
Processing derived views

Before you can use the RDML commands **GET**, **INSERT**, **UPDATE**, and **DELETE**, the derived view must open the base view. Applications do not explicitly open a base view; it is opened on first use. This may require parsing the view definition, reading a bound version of the view, or finding the view in the global view area. In any case, the view's internal data structure must be provided to execute the RDML commands. When using derived views, opening a derived view results in opening one or more base views.

For example, when you open the PRODUCTS-IN-REGION derived view, the REGION, BRANCH, STOCK, and PRODUCT base views are also opened. In combination, these views can affect every file in your physical database. After you open all of the views, you can process the RDML commands.

Processing the GET command

When you issue a **GET** for the BRANCHES-IN-REGION view (see example 2 in “**Derived views**” on page 80), RDM issues a GET for the REGION base view which causes a request to the E\$RG file. If this operation returns data for the REGION base view, RDM issues a GET for the BRANCH base view. The second GET results in a sweep of the E\$BR file searching for records with the correct region number. The following figure shows the processing sequence.



Processing the INSERT command

This example uses the PRODUCTS-IN-REGION derived view to insert a new product into the stock of a branch in a given region.

Derived View: PRODUCTS-IN-REGION

View Text:

```
> DEFINE PRODUCTS-IN-REGION
> 0100 KEY      REGION-NO
> 0200          REGION-NAME
> 0300 KEY      BRANCH-NO
> 0500          BRANCH-NAME
> 0600 KEY      STOCK-PRODUCT
> 0700          PRODUCT-DESC
> 0800 ACCESS   REGION ALLOW UPDATE DELETE
> 0900 ACCESS   BRANCH WHERE BRANCH-REGION = REGION-NO
> 1000          ALLOW ALL
> 1100 ACCESS   STOCK WHERE STOCK-BRANCH = BRANCH-NO
> 1200          ALLOW ALL
> 1300 ACCESS   PRODUCT ONCE WHERE PRODUCT-CODE = STOCK-PRODUCT
```

Because the access statements containing the REGION and PRODUCT views do not allow **INSERT**, the REGION-NO and STOCK-PRODUCT values must exist in the database before the INSERT can succeed. This view does allow for the insertion of new branches and stock into a branch without any restriction. The only reason to include the PRODUCT relation in this view is to provide the PRODUCT-DESC field. The integrity constraint between the STOCK and PRODUCT relations (no STOCK-PRODUCT number allowed which is not already in PRODUCT) is already defined in the base views.

5

Maintaining referential integrity

Referential integrity ensures that two pieces of data representing the same fact do not become inconsistent. You can set up your base views to maintain referential integrity. This chapter describes referential integrity using the following terms:

- ◆ **Foreign key.** A data field (a column or combination of columns) in one relation that can contain only values found in the primary key of another relation.
- ◆ **Primary key.** A data field (a column or combination of columns) that uniquely identifies a row in a relation. A primary key may have multiple foreign keys associated with it.
- ◆ **The source relation.** A file or relation that contains the foreign key as a data field and whose records refer to primary key values in another relation.
- ◆ **The target relation.** A file or relation that contains the primary key values that match the foreign key values in the source relation.

The terms source relation and target relation are relative and only express the relationship between two relations at a time.

The following figure shows the relationship between a source relation and a target relation. Values in *foreign-key-a* in the source relation must first exist in *primary-key-a* in the target relation.

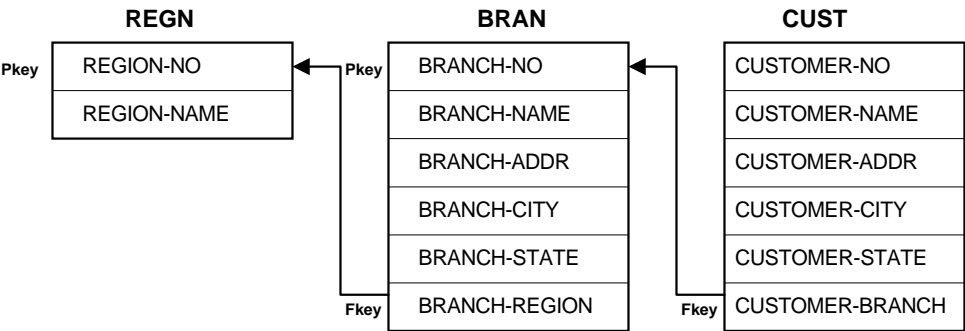
Target relation:

primary-key-a
---------------	-----	-----	-----

Source relation:

primary-key-b	...	foreign-key
---------------	-----	-------------

The examples in this chapter showing how RDM maintains referential integrity use the relations shown in the following figure.



These relations have the following foreign keys:

- ◆ CUSTOMER-BRANCH is a foreign key from CUST to BRAN. E\$CU is the source file. E\$BR is the target file.
- ◆ BRANCH-REGION is a foreign key from BRAN to REGN. E\$BR is the source file. E\$RG is the target file.

Integrity rules and checking

RDM supports the following referential integrity rules:

- ◆ A foreign key value must exist in the target relation as a primary key. A primary key value must exist for each foreign key value in a source relation.
- ◆ Null values are allowed for a foreign key.

RDM checks for referential integrity in the following ways:

- ◆ **Foreign key value integrity.** When you insert or update a record containing a foreign key, the foreign key value must point to a valid primary key in the target relation or must be null. This operation is bypassed if the foreign key is null. This rule also applies if the foreign key consists of several key parts. RDM performs **INSERT** or **UPDATE** integrity only if none of the key parts is null.
- ◆ **Deletion integrity.** RDM does not permit a record to be deleted unless you first delete or nullify all foreign keys. This means that you cannot delete a primary key unless you also delete or nullify records in a source relation that contain the key value in a foreign key.

Foreign key value integrity

To enforce foreign key value integrity, define the foreign key in the base view. You may define a required foreign key or a foreign key that allows nulls. To define a foreign key, you must:

- ◆ Make the foreign key required or identified by FKEY and redundant to the primary key in the target relation. For example:

```
REQ REGION-NO = BRANCH-REGION = REGION-NO
```

or

```
FKEY REGION-NO = BRANCH-REGION = REGION-NO
```

- ◆ Access the target relation through its primary key by using the foreign key value from a source relation. For example:

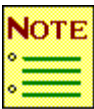
```
ACCESS E$BR ALLOW INSERT UPDATE
```

```
ACCESS E$RG ONCE WHERE REGION-NO = BRANCH-REGION
```

If you use FKEY, BRANCH-REGION must be valid or null. If you use REQ or any qualifier other than FKEY, BRANCH-REGION (the foreign key) must be valid and non-null.

The rules for defining a foreign key are:

- ◆ The foreign key may consist of one or more fields. The parts of the foreign key do not have to be contiguous in the source file. The foreign key parts must all come from the same physical file.
- ◆ You must use all the parts of the foreign key to access the target relation through its primary key. The foreign key parts must provide the full primary key. The source relation has a one-to-one or many-to-one relationship with the target relation. You must not specify additional selection criteria (using the WHERE clause) on any columns in the target relation.
- ◆ In the column definition for each column of the foreign key, the column must have a qualifier (REQ, FKEY, or other) and must be made redundant with the equivalent part of the primary key.
- ◆ Express all integrity constraints in base views. You must not use the FKEY qualifier in derived views. Refer to the *SUPRA Server PDM RDM PDM Support Supplement (OS/390 & VSE)*, P26-8221, or the *SUPRA Server PDM RDM VSAM Support Supplement (OS/390 & VSE)*, P26-8222, for information on coding base views.



A foreign key defined with REQ cannot be null.

Insertion integrity

When you attempt an insert on a relation that contains a foreign key, RDM ensures that after the insert, the foreign key points to a valid primary key in the target relation or that the foreign key is null. A foreign key can be null only if you specify FKEY in its column definition. If you insert a non-null foreign key value and the primary key in the target relation does not exist, you can have RDM perform one of two actions:

- ◆ Reject the insert. You do this by not coding ALLOW INSERT or ALLOW ALL in the access definition for the target relation. (In these examples, E\$CU is the source; E\$BR is the target.) RDM returns a column status indicator (ASI) of V for the foreign key column(s) and returns a function status indicator (FSI) of D or X. (See “[Modifying user data](#)” on page 49 for explanations of FSI values.) For example:

```
> 0100 KEY      CUSTOMER-NO
> 0200 FKEY     BRANCH-NO = CUSTOMER-BRANCH = BRANCH-NO
> 0300 ACCESS  E$CU
> 0400          ALLOW INSERT
> 0500 ACCESS  E$BR ONCE WHERE BRANCH-NO = CUSTOMER-BRANCH
```

- ◆ Automatically insert the primary key in the target relation. You do this by coding ALLOW INSERT or ALLOW ALL in the access definition for the target relation. For example:

```
> 0100 KEY      CUSTOMER-NO
> 0200 FKEY     CUSTOMER-BRANCH = CUSTOMER-BRANCH = BRANCH-NO
> 0300 ACCESS  E$CU
> 0400          ALLOW ALL
> 0500 ACCESS  E$BR ONCE WHERE BRANCH-NO = CUSTOMER-BRANCH
> 0600          ALLOW INSERT
```

If you have automatic insert of a new primary key, you can require validation of another foreign key in the automatically added row. In this case, you must also define the second foreign key. For example:

```
> 0100 KEY      CUSTOMER-NO
> 0200 FKEY     CUSTOMER-BRANCH = CUSTOMER-BRANCH = BRANCH-NO
> 0300 FKEY     REGION-NO = BRANCH-REGION = REGION-NO
> 0400 ACCESS   E$CU
> 0500          ALLOW ALL
> 0600 ACCESS   E$BR ONCE WHERE BRANCH-NO = CUSTOMER-BRANCH
> 0700          ALLOW INSERT
> 0800 ACCESS   E$RG ONCE WHERE REGION-NO = BRANCH-REGION
```

If you insert a customer record with a BRANCH-NO that does not exist, RDM inserts a branch record. However, before inserting the branch, RDM checks that REGION-NO points to an existing region record. If not, the insert fails. By placing ALLOW INSERT on the region relation, you can also make RDM perform automatic inserts on the region relation.

You can cause a foreign key to be null with an **INSERT** or **UPDATE** either by placing an N into the ASI for the foreign key column(s) or by supplying the actual null value; RDM does not perform INSERT referential integrity in this case as primary keys cannot be null. You can have a null foreign key only if you specify FKEY in the column definition in the view definition.



Cincom does not recommend inserting the actual null value because the application is then dependent on the null value.

Update integrity

When you update a foreign key, RDM ensures that after the update, the foreign keys point to a valid primary key in the target relation or that the foreign key is null. If you update the foreign key value to a non-null value and the primary key in the target relation does not exist, you can have RDM perform one of two actions:

- ◆ Reject the update. You do this by coding neither ALLOW INSERT nor ALLOW ALL in the access definition for the target relation. RDM returns a column status indicator (ASI) of V for the foreign key column(s) and returns a function status indicator (FSI) of D or X. (See “[Modifying user data](#)” on page 49 for explanations of FSI values.) For example:

```
> 0100 KEY      CUSTOMER-NO
> 0200 FKEY     CUSTOMER-BRANCH = CUSTOMER-BRANCH = BRANCH-NO
> 0300 ACCESS   E$CU
> 0400          ALLOW UPDATE
> 0500 ACCESS   E$BR ONCE WHERE BRANCH-NO = CUSTOMER-BRANCH
```

- ◆ Automatically insert the primary key in the target relation. You do this by coding ALLOW INSERT or ALLOW ALL for the target relation. For example:

```
> 0100 KEY      CUSTOMER-NO
> 0200 FKEY     CUSTOMER-BRANCH = CUSTOMER-BRANCH = BRANCH-NO
> 0300 ACCESS   E$CU
> 0400          ALLOW UPDATE
> 0500 ACCESS   E$BR WHERE BRANCH-NO = CUSTOMER-BRANCH
> 0600          ALLOW INSERT
```

If the view defines other foreign keys in the automatically inserted target relation, then insert integrity rules apply on the insertion. For example:

```
> 0100 KEY      CUSTOMER-NO
> 0200 REQ      BRANCH-REGION = BRANCH-REGION = REGION-NO
> 0300 REQ      CUSTOMER-BRANCH = CUSTOMER-BRANCH = BRANCH-NO
> 0400 ACCESS   E$CU
> 0500          ALLOW INSERT
> 0600 ACCESS   E$BR ONCE WHERE BRANCH-NO = CUSTOMER-BRANCH
> 0700          ALLOW INSERT UPDATE
> 0800 ACCESS   E$RG ONCE WHERE REGION-NO = BRANCH-REGION
```

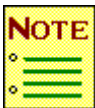
You can also specify updating on the target relation. For example, in the following view you could update both CUSTOMER-NAME and BRANCH-NAME. You could not update CUST-NO because it is a primary key, and you cannot update primary keys.

```
> 0100 KEY      CUSTOMER-NO
> 0200          CUSTOMER-NAME
> 0300          BRANCH-NAME
> 0400 REQ      BRANCH-NO = CUSTOMER-BRANCH = BRANCH-NO
> 0500 ACCESS   E$CU
> 0600          ALLOW UPDATE
> 0700 ACCESS   E$BR WHERE BRANCH-NO = CUSTOMER-BRANCH
> 0800          ALLOW UPDATE
```

In this example, if you update the foreign key field BRANCH-NO, the update processing positions the branch file on the record pointed to by the new foreign key value. This means that any update to BRANCH-NAME would apply to the branch record the new foreign key value points to, not the BRAN record retrieved by the **GET** before the update. Use care if you allow updating on both the source relation and the target relation.

You can allow both **INSERT**s and **UPDATE**s for the target relation. This means RDM can update the target relation if the primary key already exists or insert the primary key if it does not exist.

You can cause a foreign key to be null with an **INSERT** or **UPDATE** either by placing an N into the ASI for the foreign key column(s) or by supplying the literal null value; RDM does not perform INSERT referential integrity in this case as primary keys cannot be null. You can have a null foreign key only if you specify FKEY in the column definition in the view definition.



Cincom recommends using an N in the ASI rather than supplying the literal null value. N in the ASI is independent of the column's data type and independent of the column's defined null value, if any.

GET processing

If a foreign key is defined as required and redundant, a **GET** RDML command must retrieve data from both the source relation and the target relation. This means that if an existing foreign key in the database is not valid, a view with the field defined as a foreign key is unable to retrieve the bad record. RDM returns an “occurrence not found” message because required data cannot be retrieved from the target relation; that is, the source foreign-key and the target primary-key must have the same value.

In the case of a null foreign key, RDM does not perform a **GET** on the target relation because a null primary key is not allowed.

When selecting with key values, always issue the first GET command in this manner:

```
GET FIRST * USING value-1
```

Issue any subsequent GETs with the same key value in this manner:

```
GET NEXT * USING value-1
```

Whenever the selection value changes, issue the GET command in this manner:

```
GET FIRST * USING value-2
```

When you use other positional qualifiers with **GET**, such as **SAME**, **PRIOR**, or **LAST**, you may get different results for different underlying physical file types. Also, some positional keywords are not legal for some physical file types.

Deletion integrity

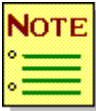
RDM does not allow you to delete a record unless you first delete or nullify all foreign keys. This means that you cannot delete a primary key if records containing foreign keys with the same value exist. To define delete referential integrity, you must access the source relation through its foreign key using the full primary key. For example:

```
> 1000 ACCESS E$RG WHERE REGION-NO = REGION-NO
> 1100          ALLOW DELETE
> 1200 ACCESS E$BR WHERE BRANCH-REGION = REGION-NO
```

If the foreign key consists of multiple parts, you must access the source relation using all its parts. For example:

```
> 1000 ACCESS E$RG WHERE REGION-NO-SUB1 = REGION-NO-SUB1
> 1100          AND REGION-NO-SUB2 = REGION-NO-SUB2
> 1200          AND REGION-NO-SUB3 = REGION-NO-SUB3
> 1300          ALLOW DELETE
> 1400 ACCESS E$BR WHERE BRANCH-REGION-SUB1 = REGION-NO-SUB1
> 1500          AND BRANCH-REGION-SUB2 = REGION-NO-SUB2
> 1600          AND BRANCH-REGION-SUB3 = REGION-NO-SUB3
```

You must not supply additional selection criteria on the WHERE clause for data fields in the source relation because RDM would use this additional criteria when checking the source relation.



For performance reasons, Cincom recommends that you index the foreign key. If the foreign key has multiple parts, include all the parts in the secondary key. An index is important because the source relation is not usually accessed through its primary key. If you try to delete a primary key, and foreign keys of the same value still exist in the source relation, you can have RDM perform one of three actions:

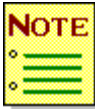
- ◆ Delete the referencing records (Cascade delete). Do this by coding ALLOW DELETE in the access definition for the source relation. If no attributes come from the source relation, RDM deletes all occurrences of the foreign key in the source relation. If attributes come from the source relation, RDM deletes only one occurrence in the source relation. RDM deletes the primary key in the target relation when the delete causes the deletion of the last referencing record.

When multiple relations depend on the source relations, RDM performs a cascade delete on all specified relations and leaves the source record if any “restrict” records exist.

- ◆ Reject the delete (Restrict delete). Do this by not coding ALLOW DELETE in the access definition for the source relation.
- ◆ Nullify the referencing foreign keys (Nullify delete) by specifying ALLOW UPDATE in the access definition for the source relation.

To enforce referential integrity during a delete operation, use one of the following options:

- ◆ Cascade delete
- ◆ Restrict delete
- ◆ Nullify delete



Cincom does not recommend that you specify Cascade delete or Nullify delete when the target relation and the source relation reside on different physical platforms (the source relation represents a SUPRA PDM file and the ref relation represents a native KSDS VSAM file). A delete operation across platforms may not be recoverable.

Cascade delete

When you perform a delete operation on a view, you must also delete all referencing rows (based on the foreign key). The following is an example of a cascade delete:

```
> 1000 ACCESS E$RG WHERE REGION-NO          = REGION-NO ALLOW DELETE
> 1100 ACCESS E$BR WHERE BRANCH-REGION      = REGION-NO ALLOW DELETE
> 1200 ACCESS E$CU WHERE CUSTOMER-BRANCH    = BRANCH-NO ALLOW DELETE
```

This example deletes a region, then all branches for the region, and all customers for the branches being deleted.

Restrict delete

A delete operation fails if any referencing rows (based on the foreign key) exist. The following is an example of a restrict delete:

```
> 1000 ACCESS REGN WHERE REGION-NO-SUB1      = REGION-NO-SUB1
> 1100                                AND REGION-NO-SUB2      = REGION-NO-SUB2
> 1200                                AND REGION-NO-SUB3      = REGION-NO-SUB3
> 1300                                ALLOW DELETE
> 1400 ACCESS BRCH WHERE BRANCH-REGION-SUB1  = REGION-NO-SUB1
> 1500                                AND BRANCH-REGION-SUB2  = REGION-NO-SUB2
> 1600                                AND BRANCH-REGION-SUB3  = REGION-NO-SUB3
```

Nullify delete

When RDM performs a delete, it deletes the primary key but nullifies the foreign key. Follow these rules to nullify a foreign key:

- ◆ Specify ALLOW UPDATE in the access definition for the source relation; you must not specify ALLOW DELETE for the source relation.
- ◆ Access the source relation joining on the foreign key and the primary key from the target relation.
- ◆ Ensure that the source relation supplies no columns.
- ◆ Specify ALLOW DELETE for the target relation.
- ◆ Set the Nulls Allowed flag for the foreign key column to Y.

The following is an example of a base view you can use to delete the primary key record and nullify the foreign key. In this example, you are deleting the region and placing null values in the BRANCH-REGION columns for the branches contained in the region. The ALLOW DELETE indicates you may delete the region. The ALLOW UPDATE on the Branch relation, BRAN, indicates you may nullify the BRANCH-REGION column.

```
> 0100 KEY      REGION-NO
> 0200          REGION-NAME
> 0300 ACCESS E$RG WHERE REGION-NO = REGION-NO
> 0400          ALLOW DELETE
> 0500 ACCESS E$BR WHERE BRANCH-REGION = REGION-NO
> 0600          ALLOW UPDATE
```

Referential integrity examples

Example 1. This view does not add a branch unless the region already exists. It does not allow updating REGION-NO in BRANCH unless the new value points to an existing region.

```
> 0100 KEY      BRANCH-NO
> 0200          BRANCH-ADDRESS
> 0300          BRANCH-CITY
> 0400          BRANCH-STATE
> 0500 REQ      REGION-NO = BRANCH-REGION = REGION-NO
> 0600 ACCESS   E$BR
> 0700          ALLOW INSERT UPDATE
> 0800 ACCESS   E$RG ONCE WHERE REGION-NO = BRANCH-REGION
```

Notice that the foreign key in the BRANCH file (E\$BR) is redundant with the primary key in the REGION file (E\$RG), and the REGION file is accessed by its primary key with the foreign key value.

Example 2. This view accesses CUSTOMER (E\$CU), then BRANCH (E\$BR), then REGION (E\$RG). It permits updates and inserts to the CUSTOMER file, and updates only to the BRANCH file. It allows no updates or deletes on the REGION file.

```
> 0100 KEY      CUSTOMER-NO
> 0200          CUSTOMER-NAME
> 0300 REQ      BRANCH-NO = CUSTOMER-BRANCH = BRANCH-NO
> 0400          BRANCH-NAME
> 0500 REQ      REGION-NO = BRANCH-REGION = REGION-NO
> 0600          REGION-NAME
> 0700 ACCESS   E$CU
> 0800          ALLOW UPDATE INSERT
> 0900 ACCESS   E$BR WHERE BRANCH-NO = CUSTOMER-BRANCH
> 1000          ALLOW UPDATE
> 1100 ACCESS   E$RG WHERE REGION-NO = BRANCH-REGION
```

An **INSERT** RDML command can insert a new customer, but to do so the column BRANCH-NO must point to an existing branch. You can **UPDATE** the customer (E\$CU) and branch (E\$BR) files. If you do an update on the column BRANCH-NO, the new foreign key value must already exist in BRANCH. Also, updating the new key value repositions the branch file before making the update to BRANCH-NAME.

Example 3. This example shows how updating a foreign key can affect the positioning of the subsequent target relations.

```
> 0100 KEY      CUSTOMER-NO
> 0200          CUSTOMER-NAME
> 0300 REQ      CUSTOMER-BRANCH = CUSTOMER-BRANCH = BRANCH-NO
> 0400          BRANCH-NAME
> 0500 REQ      BRANCH-REGION = BRANCH-REGION = REGION-NO
> 0600          REGION-NAME
> 0700 ACCESS   E$CU WHERE CUSTOMER-NO = CUSTOMER-NO
> 0800          ALLOW UPDATE
> 0900 ACCESS   E$BR WHERE BRANCH-NO = CUSTOMER-BRANCH
> 1000 ACCESS   E$RG WHERE REGION-NO = BRANCH-REGION
> 1100          ALLOW UPDATE
```

A **GET** on this example returned a row with the following column values:

```
CUSTOMER-NO      = 11111
CUSTOMER-NAME     = GEORGE
CUSTOMER-BRANCH   = 1000
BRANCH-NAME       = BRANCH 1000
BRANCH-REGION     = 100
REGION-NAME       = REGION 100
```

If the application updated the column and you issued an RDML **UPDATE** command, the result is:

```
CUSTOMER-NO      = 11111
CUSTOMER-NAME     = GEORGE WILSON
CUSTOMER-BRANCH   = 5000
BRANCH-NAME       = BRANCH CHANGE
BRANCH-REGION     = 100
```

REGION-NAME = WESTERN REGION updates to CUSTOMER-NAME and CUSTOMER-BRANCH are applied as indicated. However, the changes to CUSTOMER-BRANCH cause a repositioning of the BRANCH file to the key value of 5000.

Even though the view contains redundant foreign keys, the ALLOW phrase on the source relation controls whether you can update a foreign key. In the example, you cannot update BRANCH-REGION because there is no ALLOW UPDATE on the E\$BR file. Even though there is an ALLOW UPDATE on E\$RG, and REGION-NO is redundant in BRANCH-REGION, it does not mean you can update BRANCH-REGION.

Example 4. This view allows deleting a region if there are no associated branches:

```
> 0100 KEY      REGION-NO
> 0200          REGION-NAME
> 0300 ACCESS E$RG WHERE REGION-NO = REGION-NO
> 0400          ALLOW DELETE
> 0500 ACCESS E$BR WHERE BRANCH-REGION = REGION-NO
```

Notice that the source file BRANCH (E\$BR) is accessed through its foreign key (BRANCH-REGION) using the primary key (REGION-NO).

Example 5. This view allows deleting branches, so you can delete the region record. If no columns from the BRANCH (E\$BR) file exist in the user view, then deleting a region deletes all branches referencing the region. If there are columns from BRANCH in the user view, the program must delete each row by using a **GET DELETE** loop or by using DELETE ALL.

```
> 0100 KEY      REGION-NO
> 0200          REGION-NAME
> 0300 ACCESS E$RG WHERE REGION-NO = REGION-NO
> 0400          ALLOW DELETE
> 0500 ACCESS E$BR WHERE BRANCH-REGION = REGION-NO
> 0600          ALLOW DELETE
```

Example 6. This view is an example of combining insert, update, and delete integrity in one view. This view allows maintenance on the BRANCH (E\$BR) file. However, foreign keys restrict maintenance operations.

```
> 0100 KEY      BRANCH-NO
> 0200          BRANCH-NAME
> 0300          BRANCH-ADDR
> 0400          BRANCH-CITY
> 0500          BRANCH-STATE
> 0600 REQ      BRANCH-REGION = BRANCH-REGION = REGION-NO
> 0700 ACCESS   E$BR
> 0800          ALLOW ALL
> *            Access to E$RG is for insert integrity.
> 0900 ACCESS   E$RG WHERE REGION-NO = BRANCH-REGION
> *            Access to E$CU is for delete integrity.
> 1000 ACCESS   E$CU WHERE CUSTOMER-BRANCH = BRANCH-NO
```

You cannot perform an insert on BRANCH (E\$BR) unless the foreign key value in BRANCH-REGION already exists as a key value in the REGION (E\$RG) file. You cannot update BRANCH-REGION unless the key value already exists in the REGION file. You cannot perform a delete on BRANCH unless you first delete all referencing records in the customer file.

6

Maintaining the RDM

The physical and logical design of your database changes as data requirements change. This chapter discusses changes that impact your views and application program design, and explains how to maintain, fine-tune, and modify your RDM system to accommodate changes and to optimize performance.

Defining and testing views with DBAID

Using the DBAID utility to experiment with the various DBAID commands is a good way to learn how RDM works.

It is important to define and test your views to ensure they work correctly before putting them into production use. Using DBAID, an online and batch utility (see “[Managing views with the DBAID commands](#)” on page 133), you can define a new view without affecting the Directory, open the view, issue RDML commands, and examine the results. You can then modify the view, if necessary. A base view does not need to be defined on the Directory before a derived view can access it. A base view need only be open (not saved) to be accessed.

You can use DBAID to perform a variety of tasks: relate views to users, gather statistics on a view, save the view, and so on. As soon as you save the view, it is available to application programs, unless a bound version of the view exists (see “[View binding](#)” on page 125). You can also take existing views from the Directory, change and test them to see if they still work, all without affecting the views stored on the Directory.

Users other than the DBA can use a limited number of DBAID commands. These commands allow the application programmer to use the DBAID utility when constructing programs that use views. Programmers can use the limited DBAID commands to see how the views perform and to determine how to design the application based on the data.

The identity of the signed-on user invokes the programmer's DBAID commands. If the user is a DBA, as defined in the Directory, then DBAID recognizes all commands. However, if the signed-on user is not a DBA, only the limited DBAID commands are available. The application programmer cannot define new views or edit existing views. The programmer can access only views that are related to the programmer's (user) ID in the Directory and can access only the data available through those views.

Refer to the [SUPRA Server PDM RDM COBOL Programming Guide \(OS/390 & VSE\)](#), P26-8330, or the [SUPRA Server PDM RDM PL/1 Programming Guide \(OS/390 & VSE\)](#), P26-8331, for more information about the DBAID commands available to the application programmer.

The following examples illustrate a sample session of the DBAID utility in an online environment. You can execute the DBAID utility in a batch environment.

Signing on to DBAID and RDM

At the sign-on screen, enter your user ID and password.

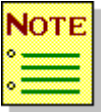
The Cincom Software Selection screen appears. Select DBAID from the list.

```
CINCOM SOFTWARE SELECTION MENU                                nnn n
ENTER SELECTION INFORMATION:
:
* TO EXECUTE WITH CURRENT USER-ID, PRESS ENTER.
* TO EXECUTE WITH ALTERNATE USER-ID, PRESS PF2/PF14

NAME      DESCRIPTION
1   DBAID      DBAID
2   NORMAL     NORMAL
3   SPECTRA    SPECTRA
4   MANTIS     MANTIS PROGRAM DEVELOPMENT
5   DIRECTRY   SUPRA ONLINE DIRECTORY MAINTENANCE
6   INTACTIV   INTERACTIVE SERVICES
7   RESIGNON   CHANGE USER-ID AND PASSWORD FOR NEXT CALL
8   CONTROL   C:M AND C:F PROGRAM PACKAGE
PF1/PF13=HELP      PA2/PA1=EXIT
```

If you do not use the Software Selection menu, you must use the **SIGN-ON** command to sign-on to RDM. The > symbolizes the system prompt. In the following examples, any data following the > is user input.

```
WELCOME TO DBAID - LEVEL nnnn
> SIGN-ON user-id password
FSI: * VSI: = MSG: SUCCESSFUL COMPLETION - LEVEL nnnn
```



If you do not use the Software Selection menu and you sign on with a user ID that begins with an asterisk (**PUBLIC**), you must enclose the user ID in single quotes.

Defining base views

This example shows the definition of a view. First, define three base views; then define a derived view using those base views.

The first base view to define is the REGION-BASE-VIEW. The numbers next to the prompt are line numbers used for editing in DBAID. The **LIST** command tells DBAID that you want to display the newly defined view. Using an asterisk (*) after a command is a shortcut and tells DBAID that you want to use the most recent *view-name* again. An * in column one of a line signifies that this is a comment line.

```
> DEFINE REGION-BASE-VIEW
> 0100 KEY      REGION-NO
> 0200          REGION-NAME
> 0300 ACCESS E$RG WHERE REGION-NO = REGION-NO ALLOW ALL
> 0400 * RESTRICT DELETION OF A REGION THAT HAS BRANCHES
> 0500 ACCESS E$BR WHERE BRANCH-REGION = REGION-NO
> LIST *
```

DBAID displays the view definition:

```
                REGION-BASE-VIEW
0100 KEY      REGION-NO
0200          REGION-NAME
0300 ACCESS E$RG WHERE REGION-NO = REGION-NO ALLOW ALL
0400 * RESTRICT DELETION OF A REGION THAT HAS BRANCHES
0500 ACCESS E$BR WHERE BRANCH-REGION = REGION-NO
```

To access the view, you must use the **OPEN** command. The FSI indicates that the OPEN was successfully completed. The VSI message indicates how much space was used in opening the view:

```
> OPEN *
FSI: *   VSI: =   MSG:                1448 BYTES USED IN OPENING VIEW.
```

Next, define BRANCH-BASE-VIEW:

```
> DEFINE BRANCH-BASE-VIEW
> 0100 KEY BRANCH-NO
> 0200     BRANCH-NAME
> 0300     BRANCH-ADDR
> 0400     BRANCH-CITY
> 0500     BRANCH-STATE
> 0600     BRANCH-ZIPCODE
> 0700     BRANCH-DEL-ROUTE
> 0800     BRANCH-SLS-QUOTA
> 0900     BRANCH-STF-QUOTA
> 1000 REQ BRANCH-REGION = BRANCH-REGION = REGION-NO
> 1100 ACCESS E$BR WHERE BRANCH-NO = BRANCH-NO ALLOW ALL
> 1200 * REJECT INSERT AND UPDATE OF BRANCH-REGION IF REGION NOT
      VALID
> 1300 ACCESS E$RG ONCE WHERE REGION-NO = BRANCH-REGION
> 1400 * REJECT DELETION OF A BRANCH THAT HAS CUSTOMERS
> 1500 ACCESS E$CU WHERE CUSTOMER-BRANCH = BRANCH-NO
> LIST *
```

The **LIST** command on the previous screen returns the following view. Remember to issue an **OPEN** command for the view.

```
BRANCH-BASE-VIEW
0100 KEY BRANCH-NO
0200     BRANCH-NAME
0300     BRANCH-ADDR
0400     BRANCH-CITY
0500     BRANCH-STATE
0600     BRANCH-ZIPCODE
0700     BRANCH-DEL-ROUTE
0800     BRANCH-SLS-QUOTA
0900     BRANCH-STF-QUOTA
1000 REQ BRANCH-REGION = BRANCH-REGION = REGION-NO
1100 ACCESS E$BR WHERE BRANCH-NO = BRANCH-NO ALLOW ALL
1200 * REJECT INSERT AND UPDATE OF BRANCH-REGION IF REGION NOT
    VALID
1300 ACCESS E$RG ONCE WHERE REGION-NO = BRANCH-REGION
1400 * REJECT DELETION OF A BRANCH THAT HAS CUSTOMERS
1500 ACCESS E$CU WHERE CUSTOMER-BRANCH = BRANCH-NO
> OPEN *
FSI: * VSI: = MSG:          4344 BYTES USED IN OPENING VIEW.
```

On the next screen, define the CUST-BASE-VIEW and **LIST** it:

```
> DEFINE CUST-BASE-VIEW
> 010 KEY CUSTOMER-NO
> 020     CUSTOMER-NAME
> 030     CUSTOMER-ADDR
> 040     CUSTOMER-CITY
> 050     CUSTOMER-STATE
> 060     CUSTOMER-ZIPCODE
> 070     CUSTOMER-CLASS
> 080     CUSTOMER-CR-CODE
> 090     CUSTOMER-CR-LIM
> 100 REQ CUSTOMER-BRANCH = CUSTOMER-BRANCH = BRANCH-NO
> 110 ACCESS E$CU WHERE CUSTOMER-NO = CUSTOMER-NO ALLOW ALL
> 120 * REJECT INSERT AND UPDATE OF CUSTOMER-BRANCH IF BRANCH
    INVALID
> 130 ACCESS E$BR ONCE WHERE BRANCH-NO = CUSTOMER-BRANCH
> LIST *
```

The **LIST** * command returns the view as entered, which must be opened in order to use it:

```
CUST-BASE-VIEW
0010 KEY CUSTOMER-NO
0020     CUSTOMER-NAME
0030     CUSTOMER-ADDR
0040     CUSTOMER-CITY
0050     CUSTOMER-STATE
0060     CUSTOMER-ZIPCODE
0070     CUSTOMER-CLASS
0080     CUSTOMER-CR-CODE
0090     CUSTOMER-CR-LIM
0100 REQ CUSTOMER-BRANCH = CUSTOMER-BRANCH = BRANCH-NO
0110 ACCESS E$CU WHERE CUSTOMER-NO = CUSTOMER-NO ALLOW ALL
0120 * REJECT INSERT AND UPDATE OF CUSTOMER-BRANCH IF BRANCH
      INVALID
0130 ACCESS E$BR ONCE WHERE BRANCH-NO = CUSTOMER-BRANCH
> OPEN *
FSI: *   VSI: =   MSG:           3952 BYTES USED IN OPENING VIEW.
```

Defining a derived view

On the next screen, design a sample derived view using the base views from above:

```
> DEFINE SAMPLE-DERIVED-VIEW
> 010 KEY CUSTOMER-NO
> 020     CUSTOMER-NAME
> 030 KEY BRANCH-NO = CUSTOMER-BRANCH = BRANCH-NO
> 040     BRANCH-NAME
> 050 KEY REGION-NO = BRANCH-REGION = REGION-NO
> 060     REGION-NAME
> 070 ACCESS CUST-BASE-VIEW WHERE CUSTOMER-NO = CUSTOMER-NO ALLOW
      ALL
> 080 ACCESS BRANCH-BASE-VIEW WHERE BRANCH-NO = CUSTOMER-BRANCH
> 090     ALLOW ALL
> 100 ACCESS REGION-BASE-VIEW WHERE REGION-NO = BRANCH-REGION
      ALLOW ALL
> LIST *
```

The **LIST *** command on the screen above returns the following view:

```
                SAMPLE-DERIVED-VIEW
0010 KEY CUSTOMER-NO
0020     CUSTOMER-NAME
0030 KEY BRANCH-NO = CUSTOMER-BRANCH = BRANCH-NO
0040     BRANCH-NAME
0050 KEY REGION-NO = BRANCH-REGION = REGION-NO
0060     REGION-NAME
0070 ACCESS CUST-BASE-VIEW WHERE CUSTOMER-NO = CUSTOMER-NO ALLOW
      ALL
0080 ACCESS BRANCH-BASE-VIEW WHERE BRANCH-NO = CUSTOMER-BRANCH
0090     ALLOW ALL
0100 ACCESS REGION-BASE-VIEW WHERE REGION-NO = BRANCH-REGION ALLOW
      ALL
```

Retrieving records

The next screen shows how to retrieve records using the SAMPLE-DERIVED-VIEW. First, issue the **OPEN** command to access the view. DBAID returns status codes and messages and the next prompt.

For this example, retrieve the first five records. Use the **GO** command to retrieve records. (To retrieve one record at a time, use the **GET** command.)

```
> OPEN *
FSI: *   VSI: =   MSG:           11060 BYTES USED IN OPENING VIEW.
> GO * FOR 5
```

The following screen shows the five records retrieved, followed by completion messages:

CUSTOMER-NO	CUSTOMER-NAME	BRANCH-NO	BRANCH-NAME	REGION-NO	REGION-NAME
D11127	ED DILLON	1264	FLORENCE	444	MID-ATLANTIC
PAID	PAID THROUGH A/R	0000	DUMMY	000	MAIN WAREHOUSE
S70703	TIM MARTIN	1273	BURNHAM	555	NEW ENGLAND
CASH	CASH TRANSACTION	0000	DUMMY	000	MAIN WAREHOUSE
S41197	JOHN ADAMS	1234	THE FARM	111	GREAT LAKES

```
FSI: *   VSI: +   MSG: SUCCESSFUL COMPLETION
```

Inserting records

To do an **INSERT** using the SAMPLE-DERIVED-VIEW, enter the INSERT command at the prompt. The example uses an * instead of the view name. DBAID prompts you for the input. If you enter any values longer than the field length, DBAID truncates the value.

```
> INSERT *
CUSTOMER-NO
> C12345
CUSTOMER-NAME
> ATLANTIS
BRANCH-NO
> 1241
BRANCH-NAME
> OAKLEY
REGION-NO
> 222
REGION-NAME
> SOUTH-EASTERN
```

When you have completed entering the record, DBAID returns a screen showing the values you keyed in, then asks if you want to do this insert. If you have entered an incorrect value, respond with N (no) at the prompt and enter the correct information. In the example, respond with a Y (yes) to complete the **INSERT**.

```
CUSTOMER-NO          ( ) C12345
CUSTOMER-NAME        ( ) ATLANTIS
BRANCH-NO            ( ) 1241
BRANCH-NAME          ( ) OAKLEY
REGION-NO            ( ) 222
REGION-NAME          ( ) SOUTH-EASTERN
INSERT (Y/N)?
> Y
```

If you have entered any incorrect values, DBAID returns the records with messages regarding their validity:

```
CUSTOMER-NO          (+) C12345
CUSTOMER-NAME        (+) ATLANTIS
BRANCH-NO            (+) 1241
BRANCH-NAME          (+) OAKLEY
REGION-NO            (+) 222
REGION-NAME          (+) SOUTH-EASTERN
```

If all entries are correct, DBAID displays a successful completion message:

```
FSI: *   VSI: +   MSG: SUCCESSFUL COMPLETION
```


Updating a row

To update a row, first issue the **GET** command to access the row in the view. In this example, you would access the first row with a key of C12345.

```
> GET * USING C12345
```

DBAID displays the row; use the **UPDATE** command to indicate that you want to modify the row.

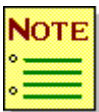
```
CUSTOMER-NO                (+) C12345
CUSTOMER-NAME              (+) ATLANTIS
BRANCH-NO                  (+) 1241
BRANCH-NAME                (+) OAKLEY
REGION-NO                  (+) 222
REGION-NAME                (+) SOUTH-EASTERN
> UPDATE *
```

For this example, change the CUSTOMER-NAME to Atlantic. At the system prompt, key in the new data. DBAID then prompts you for additional changes to BRANCH and REGION. Press ENTER at any prompts where you want to skip ahead without entering data.

```
CUSTOMER-NO
CUSTOMER-NAME
> ATLANTIC
BRANCH-NO
BRANCH-NAME
>
REGION-NO
REGION-NAME
```

> DBAID displays the row again, with the change, and asks if you want to complete the update. The FSI and VSI status codes indicate that the request was completed successfully.

```
CUSTOMER-NO                (+) C12345
CUSTOMER-NAME              (+) ATLANTIC
BRANCH-NO                  (+) 1241
BRANCH-NAME                (+) OAKLEY
REGION-NO                  (+) 222
REGION-NAME                (+) SOUTH-EASTERN
UPDATE (Y/N)?
> Y
FSI: *   VSI: +   MSG:   SUCCESSFUL COMPLETION
```



This example applies only to online DBAID. In batch DBAID you cannot skip fields you do not want to change in this manner. See the description of the **UPDATE** command in “Managing views with the DBAID commands” on page 133 for details.

Modifying a view definition

To modify a view definition, first issue the **RELEASE** command to release the open views. To display all views currently active in DBAID, use the **VIEWS** command.

For the example, modify the SAMPLE-DERIVED-VIEW.

```
> RELEASE
FSI: * VSI: = MSG: SUCCESSFUL COMPLETION
> LIST SAMPLE-DERIVED-VIEW
0010 KEY CUSTOMER-NO
0020     CUSTOMER-NAME
0030 KEY BRANCH-NO = CUSTOMER-BRANCH = BRANCH-NO
0040     BRANCH-NAME
0050 KEY REGION-NO = BRANCH-REGION = REGION-NO
0060     REGION-NAME
0070 ACCESS CUST-BASE-VIEW WHERE CUSTOMER-NO = CUSTOMER-NO ALLOW ALL
0080 ACCESS BRANCH-BASE-VIEW WHERE BRANCH-NO = CUSTOMER-BRANCH
0090     ALLOW ALL
0100 ACCESS REGION-BASE-VIEW WHERE REGION-NO = BRANCH-REGION ALLOW ALL
```

Determine which lines to change. In this example, delete line 60 and change line 90. This will delete the REGION-NAME from the view and restrict deletions.

```
> 060
> 090 ALLOW INSERT UPDATE
```

The view definition has been changed. To see how the row format was affected, issue an **OPEN** command to open the view, then issue a **GO**.

```
> OPEN *
FSI: * VSI: = MSG: 10732 BYTES USED IN OPENING VIEW.
GO * FOR 2
```

DBAID returns the first two rows according to the modified view. REGION-NAME no longer appears in the table because you deleted it.

CUSTOMER-NO	CUSTOMER-NAME	BRANCH-NO	BRANCH-NAME	REGION-NO
CASH	CASH TRANSACTION	0000	DUMMY	000
E40000	DOUG REED	1241	OAKLEY	222

```
FSI: * VSI: + MSG: SUCCESSFUL COMPLETION
```

When you have finished using DBAID, use the **BYE** command to exit:

```
> BYE
DBAID SESSION COMPLETE
```

Maintaining current programs and views

RDM insulates application programs from many changes to the physical database. However, as data requirements change and you modify the database, you may need to modify your views or application programs. These changes may require you to change your program logic and recompile the program, or modify and rebind your views.

Two RDM reports aid you in determining the impact of changes to your views and programs due to changes in your physical database implementation:

- ◆ **Views Used by Programs Report.** Shows the programs that use a particular view.
- ◆ **Impact of Change Report.** Shows the views that use a particular physical field.

With these reports, you can determine the views or programs that may be affected by a change. See [“Using the RDM reports”](#) on page 211 for more information about RDM reports.

The table on the following pages shows actions you may need to take if you make file changes, physical changes, or logical changes.

Changes to files include changing a file type. Usually, RDM insulates application programs from these types of changes while you modify and rebind the view.

Physical changes, such as changing the characteristics of a column in a view, may require a change to the program logic and recompilation. You need to recompile only the programs that use the column in their user view. You must also modify and rebind the view.

Logical changes include changing the relationships between data. For example, changing a one-to-one relationship to a one-to-many relationship usually requires that you change the program logic to process the new relationship. You must also modify and rebind the view.

If the RDM eligible flag is not set to Y on the Directory for the secondary key, an **OPEN** or a **BIND** will receive the message "Index not available to RDM."

If the RDM eligible flag is set to Y but the secondary key is not populated, an **OPEN** or a **BIND** will receive the message "Secondary Key not populated." Adding or deleting indexes, secondary keys, or linkpaths may change the behavior of unbound views. RDM selects the access strategy at view open time and adding secondary keys or indexes may alter this selection. If you want a bound view to take advantage of a newly defined index or secondary key, you must first rebind the view.

	Change program logic	Re- compile program	Modify view defn	Rebind view	Unload/ reload DB	None
File changes						
Add a new file			X	X	X	
KSDS file into PDM file			X	X	X	
PDM file to a KSDS file			X	X	X	
Combine 2 files into 1			X	X	X	
Delete a file if contains field for view	X	X	X	X	X	
Rename file			X	X	X	
Split 1 file into several			X	X	X	
Change a PDM file type			X	X		
Change a record type				X		
Change the linkpath location				X		
Change the physical key length				X		
Change the base length of a variable record				X		
Change the key position in a parent record				X		
Change a field heading				X		
Add or remove an index or secondary key				X		

	Change program logic	Re- compile program	Modify view defn	Rebind view	Unload/ reload DB	None
Physical changes						
Add new fields to a record					X	
Change Field length	X *	X	Y	X	X	
Change Field type	X *	X	Y	X		
# of decimal places	X *	X	Y	X		
Physical field's location			X	X	X	
Delete a field from physical record if used by view and program X	X	X	X	X		
Rename a physical field if field used in view			X	X		
Change edit mask/translate table			X			
Change null value or nulls allowed				X		
Change default value					X	
Change validation type				X		
Change validation data						
(Range, table name, exit)				X		
Change validation table				X		

* Does not apply to MANTIS programs.

	Change program logic	Re- compile program	Modify view defn	Rebind view	Unload/ reload DB	None
Logical changes						
Add columns and include in view				X		
Add new columns to a view			X			
External field's type	X	X	X			
External field's length	X	X	X			
Unique key to a nonunique	X		X	X	X	
Change relationships						
Program depends on relationship	X	X	X	X		
Define a new view						X
Delete a field or column			X	X		
Program uses field or column	X	X				
Move a column in a row					X	
Rename a column						
Program uses column on Include	X	X				
Reorder columns			X	X		
Other changes						
Installation of new RD Service Level Release				X		

* Does not apply to MANTIS programs.

Y If a constant column maps to the field in question.

Checking currentness of program

RDM has several checks to ensure that the program you are running is current and that the user view it uses is the same as other applications in the system. When an application program issues an RDML command, RDM checks to see if the columns in the view, as defined in the Directory, are the same as when you last compiled the program. If not, RDM returns an FSI status code, and you must recompile the program.

Application systems are often composed of several separately compiled programs that depend on common definitions of data items. These programs call each other to perform special tasks. RDM checks on each RDML call to make sure that the definition of the user view is the same for each program. If you compile a program or subroutine with the same user view name as another program or subroutine and the user view definition does not match, RDM generates an error message. The field list generated by the RDML compiler at compile time contains the data used to perform this error checking.

Checking currentness of view bindings

RDM uses the bound copy of a view whenever possible. However, RDM does not warn you if the bound view you are using is out of date. It is up to you to rebind your view whenever necessary. See the table under “[Maintaining current programs and views](#)” on page 115 for a list of reasons for rebinding a view.

Optimizing performance

This section describes techniques for optimizing your system performance:

- ◆ “[Global view support](#)” on page 122 describes global view support, which allows you to have certain views opened at system initialization, thereby improving the performance of opening the view.
- ◆ “[View binding](#)” on page 125 describes view binding, which allows you to store an opened version of a view on the Directory, thereby improving the performance of opening the view.
- ◆ “[Installing the RDM resident module in shared memory](#)” on page 126 describes storing the RDM resident module in shared memory, so that multiple applications can use the same copy and conserve program memory.

Global view support

Global view support allows you to have certain views opened during RDM initialization. By doing this, you can save RDM the processing overhead of opening views when first accessed by the application program. You define which views are global in the Directory (see the following example). A global view is available to all users related to that view.

You can make both base and derived views global. However, before you make a derived view global, you should make global each view the derived view accesses.

Batch RDM does not open global views. Global view support is available for CICS and IMS/DC users only.

Specify which views are global using Directory Maintenance. List the view names in the long text area for the Environment Description entity. Put each view name on a separate line with this syntax:

GLOBAL *view-name*

Here is an example of an Online Directory Maintenance screen with global views specified:

DIRECTORY MAINTENANCE		ENVDES: LONG EDIT
SCHEMA: DEMOSCHM		
LAST UPDATE 11.09.16. 04/01/91 V.0001 USER: THOMAS		
ENTER EDITOR COMMAND: AD SEQ1: BEG. SEQ2/INCR: 100		
0100	GLOBAL	CUSTOMERS
0200	GLOBAL	PRODUCTS
0300	GLOBAL	CUSTOMER-PRODUCTS
0400	GLOBAL	PRICES
0500	GLOBAL	CUSTOMER-POS

Refer to the *SUPRA Server PDM Directory Online User's Guide (OS/390 & VSE)*, P26-1260, or the *SUPRA Server PDM Directory Batch User's Guide (OS/390 & VSE)*, P26-1261, for detailed instructions for specifying global views with Directory Maintenance.

During initialization of RDM, RDM opens all global views in the system's memory space. For each global view, RDM displays a message on the console in the following format:

```
CSIV114I GLOBAL VIEW: view-name
FSI: * nnnn BYTES USED IN OPENING VIEW.
```

The amount of memory used to open a global view is the same as the amount of memory used to bind the view. If the view is already bound, you can use Directory reports to determine the memory requirements. The total amount used is displayed in the following format:

CSIV117I GLOBAL VIEWS OPENED; STORAGE USED IS *nnnnK* When using a global view, the user view requires less memory in the heap. The RDM user's context is one heap plus one stack; the heap and the stack are not contiguous, and the stack is only in storage for the duration of an RDML command. [“Setting the online RDM options with macros”](#) on page 261 tells how to specify the numbers and sizes of heaps and stacks. Refer to the [SUPRA Server OS/390 Installation Guide](#), P26-0149, or the [SUPRA Server VSE Installation Guide](#), P26-0132 for RDM memory requirements and usage for your operating environment. For a discussion of the relationships between RDM parameters and other SUPRA Server parameters, see [“Configuring the RDM for your environment”](#) on page 225.

The amount of reduction in heap memory is the same as the amount of memory used to open the global view. For example, suppose a user view requires 3K to open when the view is not global. The 3K are all allocated in the heap. When the view is made global, it requires 2K to open during global initialization. The 2K are allocated in the global area so when the application programmer opens the user view, only 1K are required in the heap. If multiple users open the user view, each requires 1K in their heap. They would all share the 2K allocated in the global area.

Because global views allow you to reduce the heap size, the savings in memory (because there are many heaps) may more than offset the amount of memory used by global views.

Global views also reduce the processing required to open a user view because opening the user view requires no Directory access. Even if the view is bound, global views significantly reduce the processing required to open the user view. For security checking, RDM keeps a table of global views that a task has previously been allowed to open. When opening a global view, RDM checks the table before performing directory I/Os to validate the user's ability to use the view. In many cases this means that no I/Os are required to open a global view. This can improve performance greatly when global views are repeatedly released and reopened. One table is kept per task. The table contains 32 entries and is reset at a **SIGN-ON** or **SIGN-OFF**.

You can decide which views are to be global views by determining frequency of use and size of the views. If a view requires a relatively large amount of memory and/or is accessed frequently, then include the view in the global view area. Once you open a global view, it cannot be released. You can use DBAID to create virtual copies of a global view and open and release the copies, but this does not affect application programs. No new definitions of views that are placed in the global area take effect until you shut down the RDM system and reinitialize. Refer to the *SUPRA Server PDM CICS Connector Systems Programming Guide (OS/390 & VSE)*, P26-7452, for information on commands that allow RDM to be stopped and restarted without having to cycle your CICS system.

The view you place in the global area may or may not be bound. The bound views improve performance only during the global opens performed during RDM initialization. They have no impact on opens performed for application programs. Because bound views require so much maintenance, it is usually better not to bind views that are also global.

View binding

Binding a view means translating the view definition into a form that RDM can use easily. The translated definition is the bound view. The bound view is stored in addition to the view definition text.

Binding a view reduces the processing necessary on the initial access to a view. When you open a bound view, RDM does not parse the view definition text. Cincom suggests you use view binding only for production systems and where the physical database is stable. This is because bound base views must be rebound when the physical database changes, and bound derived views must be rebound whenever the base view changes.

You can bind views by issuing the **BIND** command or the **SAVE** command in DBAID. Refer to “[Managing views with the DBAID commands](#)” on page 133 for information about DBAID commands.

You can bind existing views and rebound existing bound views with the DBAID command **BIND**:

- ◆ **BIND view-name.** Bind a view without saving it.
- ◆ **BIND BOUND.** Rebind all currently bound views.
- ◆ **BIND ALL.** Bind or rebound each view in the schema. The **BIND ALL** command takes a long time to complete and may fill your Directory because it creates bound copies of all your views.

If a bound version of a view is available, RDM applications always use it. If a bound view is not available, RDM applications use the view definition text. You can change the view definition text without affecting any applications until you rebound the view. Only after you rebound the view does it become available to application programs. To remove a view definition and a view's binding, use the DBAID command **REMOVE**. To guard against errors in using this command, you can remove only views which are also virtual views, that is, views that have been listed first. Even after you remove a view, you still have a copy of the view as a virtual view. You only lose this virtual view if you undefine the view (using the **UNDEFINE** command) or enter **BYE**, terminating the DBAID session. To keep the view, issue the **SAVE** command before terminating the session.

The DBA Report lists the time of the last binding as well as the last time the text was updated. From this report you can determine whether the view text differs from the text used in producing the binding. The End User Report and the Programmer's Report use the view definition text instead of the view bindings. It is a good practice to rebind the view whenever you change the view text.

The table under “[Maintaining current programs and views](#)” on page 115 shows which changes to your physical database or base views require you to rebind your views.

Installing the RDM resident module in shared memory

You can install the major portion of RDM, the resident module (CSVLVRES or CSVNVRES), in a shared memory area. This allows multiple users to use the same copy of the resident module and conserve memory.

Installation in the LPA under OS/390/XA

Under OS/390, you can install the resident modules into the linkpack area (LPA), but the resulting memory savings may have no practical benefit. The online resident module CSVNVRES is always loaded in extended memory (above the 16 MB line); saving extended memory is probably not significant. The batch resident module CSVLVRES is used in batch jobs, where memory is probably not at a premium.

The resident modules CSVLVRES and CSVNVRES are already linked as reentrant. To install them into the LPA, move the modules from the SUPRA Server link library into SYS1.LPA. After your next IPL, the modules reside in the linkpack area.

If you move CSVNVRES to the LPA in a CICS environment, you may require an application load table (DFHALT) entry in your CICS tables, depending on your site's requirements.

Installation in the SVA under VSE

Under VSE, you can install the resident module CSVLVRES (batch) or CSVNVRES (CICS) into the shared virtual area (SVA). These modules have already been linked using the link deck CSVLVRES or CSVNVRES and marked as shared virtual area (SVA) eligible.

If you do not want these modules to be SVA-resident, RDM loads them into the VSE GETVIS area. If you want them to be SVA-resident, load them in one of two ways:

- ◆ Add the phase name CSVLVRES or CSVNVRES to your automatic system initialization (ASI) procedure. The next time you IPL the system, CSVLVRES or CSVNVRES will be loaded into the SVA.
- ◆ Submit a VSE pause job to the background partition. Then you can load CSVLVRES or CSVNVRES into the SVA through the VSE system console.

The VSE job to implement the second method would be in the following form:

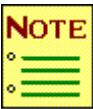
```
// DLBL TISLIB, 'dsname'
// EXTENT,nnnnnn
LIBDEF CSVNVRES, FROM=TISLIB, SEARCH=TISLIB
SET SDL
CSVLVRES, SVA
CSNVRES, SVA
/*
```

where:

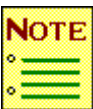
dsname Specifies the data set name of the user core image library containing the phase CSVLVRES or CSVNVRES.

nnnnnn Specifies the volume serial number of the disk containing the core image library.

SET SDL Indicates that you are building a system directory list (SDL) entry, listing phases that are SVA-eligible. CSVLVRES, SVA or CSVNVRES, SVA indicates that the resident module is to be loaded into the SVA.



The phases CSVLVRES and CSVNVRES are large. Make sure your SVA is large enough to accommodate these phases in addition to your other SVA eligible phases.



If CSVNVRES is put into the SVA, you may need to put an application load table (DFHALT) entry in your CICS tables, depending on your site requirements.

Gathering and interpreting statistics

RDM statistics track the number of RDML requests each user makes and the number of physical data manager (PDM) requests RDM makes when processing the user's RDML requests. These statistics can help indicate how efficient your views are. Under a teleprocessing monitor, only one task at a time can print statistics.

Gathering statistics with DBAID

Use the following DBAID commands to gather RDM statistics:

- ◆ **STATS-ON.** Initialize statistics to zero. Enable gathering of statistics on user views on both the logical and physical levels.
- ◆ **PRINT-STATS.** Print the current statistics to the DMLPRINT file.
- ◆ **STATS.** Display the current statistics online.
- ◆ **STATS-OFF.** Disable gathering of statistics.

See “[Managing views with the DBAID commands](#)” on page 133 for specific information about each DBAID command.

Gathering statistics in an application program

You can gather statistics from your application program. To do this, move one of the following 4-byte codes into the TIS-OPTIONS field in the TIS-CONTROL-AREA before the RDML call:

- ◆ SSTA Equivalent to the **STATS-ON** command
- ◆ ESTA Equivalent to the **STATS-OFF** command
- ◆ PSTA Equivalent to the **PRINT-STATS** command

Interpreting RDM statistics

RDM prints statistics in a tabular report format. The first part of the report is a table showing all the open user views and the number of RDML requests made by the task.

The second part of the report contains a table showing the user view name and the view used. The table shows which files the view contained. There is one line in the table for each ACCESS statement in the view. Each line shows how many requests to the physical data manager were performed.

There are no hard and fast rules for interpreting RDM statistics. Statistics vary depending on whether you are penetrating a file or sweeping it. When you sweep a file, there are more RDM calls to the physical data manager than when you supply a key value and penetrate a file.

Statistics example

Following is an example showing the statistics gathered during the sample DBAID session in “[Defining and testing views with DBAID](#)” on page 104:

```
> STATS SAMPLE-DERIVED-VIEW
      VIEW NAME          GET      INSERT      UPDATE      DELETE
      SAMPLE-DERIVED-VIEW          7          1          0          0

LVL      ACCESS NAME
0 | CUST-BASE-VIEW          7          1          0          0
0 | BRANCH-BASE-VIEW        6          0          0          0
0 | REGION-BASE-VIEW        6          0          0          0

> STATS-OFF
```

The statistics report is in two parts. The first part of the report shows that seven logical **GET**s were performed on the user view, SAMPLE-DERIVED-VIEW.

The second part of the report gives the name of the user view and the views used. In this example, when the CUST-BASE-VIEW was used to **GET** seven records, there were seven reads to the CUST-BASE-VIEW, six reads to the BRANCH-BASE-VIEW, and six reads to the REGION-BASE-VIEW. One customer record was inserted.

LVL refers to the level of occurrence for the column name. (See the description of the **BY-LEVEL** command in “[Managing views with the DBAID commands](#)” on page 133 for more details.)

Relating views to users

You control the security of the database by defining on the Directory which users can use which views are related to which users. A user can use a view only if that view is related to that user or that view is related to the ****PUBLIC**** user.

You can relate a view to a user with one of the following:

- ◆ The DBAID command **PERMIT**
- ◆ The DBAID command **PUBLIC-PERMIT** (to relate a view to the ****PUBLIC**** user)
- ◆ The Directory Maintenance **RELATE** command

Whether you use DBAID or Directory Maintenance to create the relationship, the relationship is stored on the Directory.

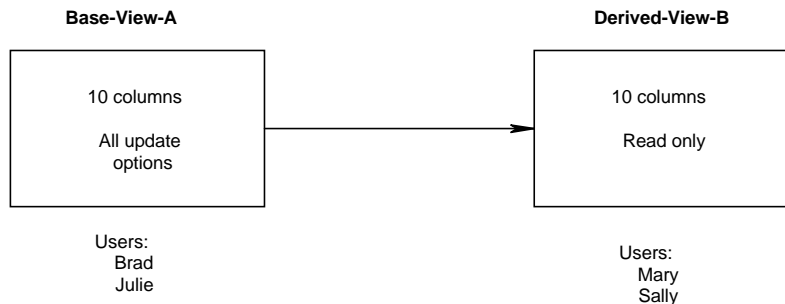
You can remove the relationship of a view to a user with one of the following:

- ◆ The DBAID command **DENY**
- ◆ The DBAID command **PUBLIC-DENY** (to remove a view's relationship to the ****PUBLIC**** user)
- ◆ The Directory Maintenance **REMOVE** command

For information about the DBAID commands, see “[Managing views with the DBAID commands](#)” on page 133. For information about the Directory Maintenance commands, refer to the [SUPRA Server PDM Directory Online User's Guide \(OS/390 & VSE\)](#), P26-1260, or the [SUPRA Server PDM Directory Batch User's Guide \(OS/390 & VSE\)](#), P26-1261.

You need not use the same program to remove a relationship that you used to create it. You can remove a relationship with DBAID whether you created it with DBAID or not. You can remove a relationship with Directory Maintenance whether you created it with Directory Maintenance or not.

You can relate both base and derived views to users. However, you can relate users to a derived view without authorizing them to use the base view that the derived view accesses. While the derived view accesses the base view, it can impose additional security on the user. The following figure shows BASE-VIEW-A which has all update capabilities. Brad is related to BASE-VIEW-A while Mary is related to DERIVED-VIEW-B which accesses BASE-VIEW-A. Mary cannot directly use BASE-VIEW-A. DERIVED-VIEW-B provides additional security, and Mary only has read access to the information contained in BASE-VIEW-A.



Recovering data

RDM itself provides no recovery. You can recover the data in a physical file if your physical file manager, teleprocessing monitor, or other program provides the ability to recover that file. For example, the SUPRA physical data manager (PDM) provides the ability to recover native PDM files. CICS provides the ability to recover certain files you define in your CICS tables.

If you update data on different physical platforms (SUPRA PDM, native KSDS VSAM) in the same logical unit of work, that set of updates may not be recoverable as a whole. Partial recovery of a logical unit of work leaves your data logically inconsistent.

RDM supports access to KSDS VSAM files, but neither SUPRA Server nor VSAM provides the ability to recover such files.

When the SUPRA PDM is running with Task Level Recovery (TLR), the RDML **COMMIT** command, issued by DBAID or by an application, makes all updates to the PDM database permanent. The RDML **RESET** command backs out any database updates since the last COMMIT.

CICS provides Dynamic Transaction Backout (DTB). A **COMMIT** makes all updates to the database permanent and takes a CICS sync point. A **RESET** backs out any database updates since the last COMMIT but does not restart the task. Instead, the RESET command performs the CICS rollback operation and then returns control to the program.

For information on logging and recovery in SUPRA Server, refer to the *SUPRA Server PDM Logging and Recovery Guide (OS/390 & VSE)*, P26-2223.

Managing views with the DBAID commands

This chapter introduces the DBAID utility and describes each of the DBAID commands in alphabetical order.

Introduction to DBAID

With the DBAID utility you can manage views. DBAID provides you with functions that include the following:

- ◆ Create, edit, display, open, bind, close, and delete views.
- ◆ Store views on the Directory.
- ◆ Remove views from the Directory.
- ◆ Create, modify, and remove relationships between views and users on the Directory.
- ◆ Add, read, update, and delete user data records with views.
- ◆ Save position information for user data records.
- ◆ Finalize (commit) or reverse (reset) a series of operations to user data records.
- ◆ Set parameters for DBAID output.
- ◆ Enable, initialize, and disable statistics gathering.
- ◆ Display information relevant to view management, including statistics, list of views, list of public views, list of users, list of views for a user, and list of users for a view.

You can define and manage both base views and derived views with DBAID. For information on defining derived views, see “[Defining and using derived views](#)” on page 63. For information on defining base views, see the SUPRA RDM support supplement(s) for the physical data platform(s) you use:

- ◆ *SUPRA Server PDM RDM PDM Support Supplement (OS/390 & VSE), P26-8221*
- ◆ *SUPRA Server PDM RDM VSAM Support Supplement (OS/390 & VSE), P26-8222*

RDM and DBAID provide limited support for positional keywords (SAME, PRIOR, LAST, etc.) for the [GET](#), [GO](#), and [INSERT](#) commands depending on the physical data platform. The support is limited because of differences in the platforms and because record position is not a relational concept. Some positional keywords RDM does not accept for some platforms. You may get different results for the same keyword for different platforms. For details, see the SUPRA Server RDM support supplement(s) for the platform(s) you use.

There are two versions of DBAID: batch DBAID and online DBAID. The format of the commands are identical in the two versions. The effects of the commands are identical in the two versions except for the [UPDATE](#) command; see the [UPDATE](#) command considerations for details.

Your SUPRA Server libraries contain procedures and job control language (JCL) samples for running batch DBAID. Samples are subject to change. See the SUPRA Server JCL library or source statement library member TXJ\$INDX for a list of JCL samples.

OS/390

See the SUPRA Server procedure library member TIS\$RDM for a list of RDM procedures. See the SUPRA Server macro library or source statement library member TX\$\$INDX for an index to the different kinds of samples. For more information on JCL samples, refer to the *[SUPRA Server PDM and Directory Administration Guide \(OS/390 & VSE\)](#)*, P26-2250.

Each DBAID command must be coded on a single line or input record. The command and its operands must fit in the first 72 columns of its line. A line with an asterisk in the first column is a comment; DBAID ignores comment lines. DBAID commands are divided into the following categories:

- ◆ **System commands.** Use these to display information about the DBAID utility currently executing. They display information such as current users and active views, and they perform functions on the Directory for you.
- ◆ **Editing commands.** Use these to change existing view definitions, as well as to create new views to test before saving them on the Directory.
- ◆ **RDML commands.** Use these to test data with a defined view to make sure the view is properly defined.
- ◆ **Built-in view commands.** Use these to inspect the view after it is opened.
- ◆ **Statistics commands.** Use these to gather, display, and print statistics on a particular user area. The following tables list all the commands by category and gives a brief description.

System commands

Command	Description
BIND	Binds a view.
COPY	Copies the view definition of one view to another view. Only the DBA can use this command.
DENY	Removes the relationship between a user and a view on the Directory.
LINESIZE	Specifies width of line for DBAID output.
MARKS	Lists all open MARKs and the views they are marking.
PAGESIZE	Specifies the number of lines on the page/screen for DBAID output.
PERMIT	Relates a view to a user on the Directory.
PUBLIC-DENY	Removes the relationship between the view entities and the **PUBLIC** user on the Directory.
PUBLIC-PERMIT	Relates a view(s) to the **PUBLIC** user on the Directory.
PUBLIC-VIEWS	Lists the names and short text for the views related to the **PUBLIC** user.
REMOVE	Removes the view definition, view bindings, and the relationship between the view and the schema. Only the DBA can use this command.
SHOW-NAVIGATION	Displays the access strategies RDM uses for entities (files/views) accessed in open views.
USER-LIST	Displays column list for the view named.
VIEWS	Displays all views active in DBAID.

Editing commands

Command	Description
DEFINE	Defines a name for a virtual view. Only the DBA can use this command.
EDIT	Readies a stored or virtual view for modification. Only the DBA can use this command.
<i>line-number</i>	Deletes, adds, or replaces a line in the view you are currently editing. Only the DBA can use this command.
LIST	Lists a stored or virtual view and readies it for modification. Only the DBA can use this command.
RENUMBER	Renumbers a virtual view so that line numbering starts at ten with each line incremented by ten. Only the DBA can use this command.
UNDEFINE	Deletes a defined virtual view.

RDML commands

Command	Description
=	Reissues the previous RDML command.
BYE	Exits the DBAID utility.
CAUTIOUS	Prohibits an automatic COMMIT.
COMMIT	Makes all updates since last commit permanent in the database.
DELETE	Removes a row from the database.
ERASE	Issues an RDM RESET if an X FSI is returned.
FORGET	Frees the storage allocated by a previously issued MARK command.
GET	Retrieves and displays the requested row for the indicated view.
GO	Issues multiple GET commands, and displays the rows in tabular format.
INSERT	Places a row in the physical database based on relative location specified.
KEEP	Prohibits an automatic RESET.
MARK	Marks the current position of the view established by the previous GET.
OPEN	Readies either a virtual or stored view for use by the DBAID utility.
RELEASE	Closes one or all views that have been opened, and releases the occupied storage.
RESET	Forces a task level abend and rolls back any database updates since the last commit.
SIGN-OFF	Signs off the user from the DBAID utility.
SIGN-ON	Identifies the user to the DBAID utility.
SURE	Causes a COMMIT after each successful insert, update, or delete.
UPDATE	Updates data values in the database.

Built-in view commands

Command	Description
BY-LEVEL	Displays the column names in the view by level of occurrence.
COLUMN-DEFN	Displays the full description of a column in a view.
COLUMN-TEXT	Displays the short and long text for a column in a view.
FIELD-DEFN	Displays a description of a column in a view. This information is a subset of the information returned by COLUMN-DEFN.
VIEW-DEFN	Displays a condensed description of the view.
VIEWS-FOR-USER	Lists the views related to the signed-on user and the short text for the view.

Statistic commands

Command	Description
PRINT-STATS	Prints the current statistics. Only the DBA can use this command.
STATS	Displays the current statistics for all open views or a particular open view, online. Only the DBA can use this command.
STATS-OFF	Prints the current statistics and then disables the statistics gathering. Only the DBA can use this command.
STATS-ON	Initializes statistics to zero and then enables the statistics gathering on user views on both the logical and physical levels. Only the DBA can use this command.

= command

The = command reissues the previous RDML command.

=

Example

In the following example, = causes another “GET NEXT CUST-PROD-VIEW.”

```
GET NEXT CUST-PROD-VIEW
=
```

BIND command

The BIND command binds or rebinds view(s) that are stored on the Directory. If there are two versions of a view, a virtual version and a saved version, BIND will bind the saved version without saving or affecting the virtual version. The DBA can use the BIND command to bind a view without also saving it (BIND *view-name*). The DBA may rebind all currently bound views (BIND BOUND) or bind all the views in a schema (BIND ALL).

BIND [*schema-name*:]

*

ALL

BOUND

view - name

schema-name:

- Description

Optional. Identifies the schema in which the view is to be bound.
- Format

Must be a valid schema defined on the Directory. The schema name must be followed by a colon.
- Consideration

If you omit this parameter, DBAID uses the active schema.

*

ALL

BOUND

view - name

- Description

Required. Specifies which views to bind.
- Format

View-name must be the name of an existing saved view.
- Options

*

Bind the view you used most recently.

ALL

Bind all views in the schema.

BOUND

Rebind all currently bound views in the schema.

view-name

Bind the named view.

General considerations

- ◆ The BIND ALL command takes a long time to complete and may fill your Directory, because it binds each of your views.
- ◆ After successfully binding each view, DBAID issues a **COMMIT**.
- ◆ If an error occurs while binding a view, DBAID issues a **RESET** for that view and continues processing with any remaining views.
- ◆ You can use this command only if your active environment description specifies update access to the Directory. For information on maintaining your environment description, refer to the *SUPRA Server PDM Directory Online User's Guide (OS/390 & VSE)*, P26-1260, or the *SUPRA Server PDM Directory Batch User's Guide (OS/390 & VSE)*, P26-1261.
- ◆ Binding a derived view has no effect on the views it accesses.
- ◆ You must rebind a derived view after changing a view it accesses.

Example

This example binds BRANCH-VIEW. Notice the message indicating how many bytes were used.

```
> BIND BRANCH-VIEW
BINDING BRANCH-VIEW
FSI: *   VSI: = MSG:                2504 BYTES USED IN VIEW BINDING.
VIEW BINDING SUCCESSFUL
```

BYE command

The BYE command exits the DBAID utility.

BYE

General considerations

- ◆ In an online environment, the BYE command returns you to the RDM signon screen or other user-installed menu screens.
- ◆ In a batch environment, the BYE command terminates the task.
- ◆ The BYE command erases all unsaved virtual views. Because virtual views are stored in memory, only those explicitly saved will be stored on the Directory. Be sure to **SAVE** any new or altered view definitions on the Directory before leaving DBAID if you want to keep them.
- ◆ If you entered DBAID with the task already signed-on to RDM, the BYE command does not perform a **SIGN-OFF**. If you entered DBAID with the task signed-off from RDM, which requires you to issue a **SIGN-ON**, the BYE command performs a SIGN-OFF.
- ◆ The BYE command prints statistics and then disables them if statistics are on.

BY-LEVEL command

The BY-LEVEL command displays the column names in a view by level of occurrence starting with level 0, followed by level 1, and so on. RDM generates the column number when displaying this data.

BY-LEVEL $\left[\begin{matrix} * \\ \textit{view - name} \end{matrix} \right] [\textit{column-number}]$

$\left[\begin{matrix} * \\ \textit{view - name} \end{matrix} \right]$

Description	<i>Optional.</i> Specifies the one view whose column names you want to display.	
Format	<i>View-name</i> must be the name of an existing, opened view.	
Options	*	Display column names for the view you used most recently.
	<i>view-name</i>	Display column names for the named view.
Consideration	If you omit this parameter, BY-LEVEL displays all column names for all your opened views.	

column-number

Description	<i>Optional.</i> The number of the one column whose name you want to display.	
Format	Numeric characters.	
Considerations		
	◆	To use this parameter, you must have specified a view.
Considerations		
	◆	If you omit this parameter, the BY-LEVEL command displays all column names of the indicated view(s).

Example

> BY-LEVEL			
NUMBER	VIEW NAME	FIELD NAME	LEVEL
1	REGN	REGION-NO	0
2	REGN	REGION-NAME	0
1	BRANCH-VIEW	BRANCH-NO	0
2	BRANCH-VIEW	BRANCH-NAME	0
3	BRANCH-VIEW	BRANCH-ADDR	0
4	BRANCH-VIEW	BRANCH-CITY	0
5	BRANCH-VIEW	BRANCH-STATE	0
6	BRANCH-VIEW	BRANCH-ZIPCODE	0
7	BRANCH-VIEW	BRANCH-REGION	0

CAUTIOUS Command

The CAUTIOUS command disables the DBAID automatic COMMIT facility. This command is the opposite of the SURE command. When you use CAUTIOUS, DBAID does not automatically issue a COMMIT when an RDML INSERT, UPDATE, or DELETE command returns an “*” FSI. Instead, you must issue the COMMIT.

CAUTIOUS

General considerations

- ◆ DBAID normally issues a COMMIT after every successful RDML modification. The CAUTIOUS command is not required; however, you can use it when you want manual control over COMMIT commands when updating the database.
- ◆ CAUTIOUS does not affect the COMMIT that system commands (REMOVE, SAVE, BIND, PERMIT, and DENY) may issue. These COMMITs must be issued after you modify the Directory.

COLUMN-DEFN command

The COLUMN-DEFN command displays the full internal description of columns in a view.

```
COLUMN-DEFN [*
view - name] [column-name]
```

```
[*
view - name]
```

- Description** *Optional.* Specifies the one view whose column descriptions you want to display.
- Format** *View-name* must be the name of an existing, opened view.
- Options**

*** Display column descriptions for the view you used most recently.

view-name Display column descriptions for the named view.
- Consideration** If you omit this parameter, COLUMN-DEFN displays all column descriptions for all your opened views.

column-name

- Description** *Optional.* Identifies the one column whose description you want to display.
- Format** The name of an existing column in the specified view.
- Considerations**

◆ If you use this parameter, you must have specified a view.

◆ If you omit this parameter, the COLUMN-DEFN command displays all column descriptions for the indicated view(s).
- 146

P26-8220-64

General consideration

The information returned by **FIELD-DEFN** is a subset of the information returned by COLUMN-DEFN.

Example

This example shows a description of one of the columns in the BRAN view.

```
> COLUMN-DEFN
VIEW-NAME                (+) BRAN
COL-NAME                  (+) BRANCH-NO
COL-POS                   (+) 0
COL-LEN                   (+) 4
COL-ASI-POS               (+) 83
COL-DEC                   (+) 0
COL-OUTP-LEN              (+) 4
COL-MASK-LEN              (-) 0
COL-FORMAT                (+) C
COL-MASK                  (-)
COL-HEADING               (-)
COL-DEL-OPT               (+) Y
COL-INS-OPT               (+) Y
COL-UPD-OPT               (+) N
COL-REDUND                (+) N
COL-CONSTANT              (+) N
COL-LEVEL                 (+) 0
COL-KEY-NUM               (+) 1
COL-REQUIRED              (+) Y
COL-UNIQUE                (+) Y
COL-EDIT-TRANS            (+)
COL-ORDERING              (-)
COL-SIGNED                (+) N
COL-NULLS-OK              (+) N
COL-NULL-LEN              (-) 0
COL-NULL-VAL              (-)
COL-DOMAIN                (+) DM-BRANCH-IDENTIFIERS
COL-VAL-TYP               (+)
COL-GET-VAL               (+) N
COL-MIN-LEN               (+) 0
COL-MIN-VAL               (-)
COL-MAX-LEN               (-) 0
COL-MAX-VAL               (-)
COL-VAL-TABLE             (-)
COL-EXIT                  (+) -----
COL-SRC-TYP               (+) F
COL-SRC-COL               (+) BRANCH-NO
COL-SRC-REL               (+) E$BRCTRL
COL-INT-REL               (+) E$BR
COL-RC                    (+)
```

COLUMN-TEXT command

The COLUMN-TEXT command displays the short and long text for a column in a view. For compatibility purposes, you can use the FIELD-TEXT command in the same manner as the COLUMN-TEXT command.

COLUMN-TEXT [^{*}*view - name*] [*column-name*]

[^{*}*view - name*]

- Description** *Optional.* Specifies the one view whose column text you want to display.
- Format** *View-name* must be the name of an existing, opened view.
- Options**

***Display column text for the view you used most recently.

*view-name*Display column text for the named view.
- Consideration** If you omit this parameter, COLUMN-TEXT displays the text for each column in each of your opened views.

column-name

- Description** *Optional.* Identifies the one column for which you want to display the short and long text.
- Format** The name of an existing column in the specified view.
- Considerations**

◆ If you use this parameter, you must have specified a view.

◆ If you omit this parameter, the COLUMN-TEXT command displays the text for each column in the indicated view(s).

Example This example shows the short and long text describing the BRANCH-ADDR column in the BRAN view.

```
> COLUMN-TEXT BRAN BRANCH-ADDR
VIEW NAME          COLUMN NAME    SHORT TEXT    LONG TEXT
BRAN               BRANCH-ADDR
                   BRANCH STREET ADDRESS
```

COMMIT command

The COMMIT command makes permanent in the database all updates since the last COMMIT (a logical unit of work).

COMMIT

General considerations

- ◆ DBAID issues a COMMIT after every successful RDML modification unless you have issued a **CAUTIOUS** command. You can use the COMMIT command to issue a COMMIT if you have issued a CAUTIOUS command.
- ◆ The system commands **REMOVE**, **SAVE**, **BIND**, **PERMIT**, and **DENY** issue a COMMIT after successful modification of the Directory.
- ◆ The physical action RDM performs for COMMIT, if any, depends on the physical platform and operating environment.

COPY command

The COPY command creates a new view with the same definition as an existing view.

COPY [*schema-name*:] *view-name*₁ *view-name*₂

schema-name:

- Description** *Optional.* Identifies the schema where the view to be copied is defined.
- Format** Must be a valid schema defined on the Directory. The schema name must be followed by a colon.
- Consideration** If you omit this parameter, COPY uses the active schema.

view-name₁

- Description** *Required.* Identifies the name of the view to copy.
- Format** Must be a valid view on the Directory for the schema.

view-name₂

- Description** *Required.* Identifies the new name for the view being copied.
- Format** 1–30 alphanumeric characters and the special characters # and \$. The first character must be alphabetic or a special character. If the first character is a special character, the second character must be alphabetic.
- Consideration** After copying, DBAID lists the new view (see the **LIST** command) and makes it available for editing.

General consideration

When DBAID performs the COPY command, it first searches for a virtual view with the name of *view-name₁*. If it does not find this view, DBAID searches the Directory for the view. Once it finds the view on the Directory, it creates a virtual view of *view-name₁*. Finally, it copies the view definition of *view-name₁* to *view-name₂*, and lists *view-name₂*.

Examples

- ◆ The following example copies CUSTOMER from the Directory for the active schema and names it NEW-CUSTOMER. DBAID lists NEW-CUSTOMER and makes it available for editing.

```
> COPY CUSTOMER NEW-CUSTOMER
```

- ◆ The following example copies TEST-VIEW from the Directory for the schema SCHEMAXX and names it PRODUCTION-VIEW:

```
> COPY SCHEMAXX: TEST-VIEW PRODUCTION-VIEW
```

DEFINE command

The DEFINE command defines a new view name to DBAID.

DEFINE *view-name*

view-name

- | | |
|--------------------|--|
| Description | <i>Required.</i> Specifies the name to use as a new view name. |
| Format | 1–30 alphanumeric characters and the special characters #, -, and \$. The first character must be alphabetic or a special character. If the first character is a special character, the second character must be alphabetic. |

General considerations

- ◆ The DEFINE command does not go to the Directory to retrieve a view. It creates a virtual view, one that exists only within the DBAID execution. You can also save a virtual view in the Directory (see the **SAVE** command).
- ◆ Once you have issued the DEFINE command, you can use the *line-number* command to begin creating your view.
- ◆ DBAID terminates DEFINE when it encounters a command other than the *line-number* command.
- ◆ You can change previously defined views with the **EDIT** command or list them with the **LIST** command.
- ◆ Remove a defined view from DBAID with the **UNDEFINE** command. However, if the defined view is open, an UNDEFINE command does not release the RDM view definition.

DELETE command

The DELETE command removes a row from the database.

DELETE [ALL] {^{*}
view – name}

ALL

Description *Optional.* Deletes all rows retrieved by automatically generated GET NEXTs using the logical key specified on the GET command.

Consideration If a program specifies a **GET** without a USING phrase, DELETE ALL deletes all rows in the relation. DELETE ALL can have far reaching effects, so be sure that the prior GET call limits the DELETE to the desired rows.

{^{*}
view – name}

Description *Required.* Specifies the name of the view for the relation containing the row(s) to delete.

Format *View-name* must be the name of an existing, opened view.

Options * Delete rows for the view you used most recently.

view-name Delete rows for the named view.

General considerations

- ◆ Before performing the DELETE, you must perform a successful **GET** command.
- ◆ Any ASIs resulting from a DELETE have no meaning.

Examples

- ◆ This example deletes the one row of SAMPLE-VIEW that was obtained based on the value in KEY1.

```
> GET SAMPLE-VIEW USING KEY1  
> DELETE SAMPLE-VIEW
```

- ◆ This example deletes all rows in SAMPLE-VIEW:

```
> GET SAMPLE-VIEW USING KEY1  
> DELETE SAMPLE-VIEW
```

The preceding two DBAID commands have the same effect as the following COBOL code:

```
RETURN.  
  GET NEXT SAMPLE-VIEW FOR UPDATE USING KEY1  
  NOT FOUND GO TO CONTINUE.  
  DELETE SAMPLE-VIEW.  
  GO TO RETURN.  
CONTINUE.
```

DENY command

The DENY command revokes a user's privilege to use a view. The command removes the relation between the user and the view entities on the Directory. This command provides security because it allows the DBA to define in the Directory who can use a view.

$$\text{DENY} \left\{ \begin{array}{c} * \\ \text{view} - \text{name} \end{array} \right\} \text{user} - \text{name}_1 \left[\left\{ \begin{array}{c} \text{b} \\ , \end{array} \right\} \text{user} - \text{name}_{2...} \right]$$

$$\left\{ \begin{array}{c} * \\ \text{view} - \text{name} \end{array} \right\}$$

Description *Required.* Specifies the view for which you are denying the user access.

Format *View-name* must be the name of an existing view.

Options * Deny access to the view you used most recently.
 view-name Deny access to the named view.

$$\text{user} - \text{name}_1 \left[\left\{ \begin{array}{c} \text{b} \\ , \end{array} \right\} \text{user} - \text{name}_{2...} \right]$$

Description *Required.* The name(s) of the user(s) you are denying access to the view.

Format Each *user-name* must be the name of an existing user defined on the Directory. User names must be separated by a comma or a blank.

General considerations

- ◆ You can use the DENY command to remove the relationship between a user and a view, regardless of whether you created the relationship with Directory Maintenance or the **PERMIT** command.
- ◆ After successfully removing the view's relationship with each user, DBAID commits the Directory update.
- ◆ If an error occurs while removing the relationship between the view and a user, DBAID backs out (resets) the Directory update and terminates processing of the command.
- ◆ You can use this command only if your active environment description specifies update access to the Directory.

EDIT command

The EDIT command readies a saved or virtual view for modification.

```
EDIT [schema-name:] { *  
                      [view – name] }
```

schema-name:

Description	<i>Optional.</i> Identifies the schema in which the view to be edited is defined.
--------------------	---

```
{ *  
  [view – name] }
```

Description	<i>Required.</i> Identifies the view to be edited.
Format	<i>View-name</i> must be the name of an existing view.
Options	<div><div>*</div><div>Edit the view you used most recently.</div></div> <div><div><i>view-name</i></div><div>Edit the named view.</div></div>

General considerations

- ◆ When you issue the EDIT command, the system first searches for a virtual view. If it does not find it, the system then searches the Directory.
- ◆ Once you have issued the EDIT command, you can use the *line-number* command to modify your view. When you enter a command other than a LINE-NUMBER command, DBAID terminates the EDIT.
- ◆ You enter the EDIT mode automatically after a *LIST* or *DEFINE* command.
- ◆ The *LIST* command can display a view before or after editing. The *LIST* command automatically issues an EDIT.
- ◆ EDIT changes exist only within the DBAID execution, but you can save them on the Directory with the *SAVE* command.

ERASE command

The ERASE command causes DBAID to issue a RDM **RESET** if an RDML command returns an X FSI. This command is the opposite of the **KEEP** command.

ERASE

FIELD-DEFN command

The FIELD-DEFN command displays the full description(s) of column(s) in a view.

```
FIELD-DEFN [*
view - name] [column-name]
```

```
[*
view - name]
```

Description	Optional. Specifies the one view whose column descriptions you want to display.	
Format	View-name must be the name of an existing, opened view.	
Options	*	Display columns descriptions for the view you used most recently.
	view-name	Display columns descriptions for the named view.
Consideration	If you omit this parameter, FIELD-DEFN displays all column descriptions for all your opened views.	

column-name

Description	Optional. Identifies the one column whose description you want to display.	
Format	The name of an existing column in the specified view.	
Considerations		
	◆	If you use this parameter, you must have specified a view.
Considerations		
	◆	If you omit this parameter, the FIELD-DEFN command displays all column descriptions for the indicated view(s).

General consideration

The information returned by FIELD-DEFN is a subset of the information returned by **COLUMN-DEFN**.

Example

This example shows a description of all the columns in your opened views.

```
> COLUMN-DEFN
VIEW-NAME                (+) CUSTOMER
FIELD-NAME                (+) CUSTOMER-NO
FIELD-POS                (+) 0
FIELD-LEN                (+) 6
ASI-POS                  (+) 83
FIELD-DEC                (+) 0
OUTPUT-LEN               (+) 6
MASK-LEN                 (-) 0
FORMAT                   (+) C
EDIT-MASK                (-)
HEADING                  (-)
DELETABLE                (+) Y
INSERTABLE               (+) Y
REPLACEABLE              (+) N
FIELD-LVL                (+) 0
KEY-NUMBER               (+) 1
REQUIRED                 (+) Y
UNIQUE                   (+) Y
EDIT-TRANS               (+)
ORDERING                 (-)
SIGNED                   (+) N
***MORE***
```

FORGET command

The FORGET command frees the storage allocated by a previously issued **MARK** command.

FORGET *mark-name*

mark-name

Description *Required.* Specifies what mark information should be forgotten.

Format An existing mark name.

Consideration Must be a name you assigned with the **MARK** command.

General consideration

Once you have issued a FORGET command, DBAID releases the indicated mark and you cannot regain it without issuing a new **MARK** command.

GET command

The GET command retrieves and displays a row for the indicated view.

```

GET [NEXT
    LAST
    SAME
    FIRST
    PRIOR] { *
          view - name } [FOR UPDATE] [AT mark - name
                                   USING literal1 literal2 ...]
  
```

NEXT
LAST
SAME
FIRST
PRIOR

Description *Optional.* Modifies the order of row retrieval.

Default NEXT If no current position exists, NEXT defaults to FIRST.

Considerations

- ◆ For a unique key:
 - GET NEXT Retrieves either the row immediately after the current row or the first row if no current position exists.
 - GET LAST Retrieves the last row.
 - GET SAME Retrieves the latest row if a current position exists.
 - GET FIRST Retrieves the first row.
 - GET PRIOR Retrieves either the row immediately before the current row or the last row if no current position exists.
- ◆ For a nonunique key:
 - GET NEXT Retrieves the next row within the generic group.
 - GET LAST Retrieves the last row.
 - GET SAME Retrieves the latest row if a current position exists.
 - GET FIRST Retrieves the first row with the indicated key.
 - GET PRIOR Retrieves the previous row within the group of nonuniquely keyed rows.
- ◆ The effect of the positional keywords varies depending on the physical data platform. See the discussion of [positional keywords](#) at the beginning of this chapter.

**{ *
view - name }**

Description	<i>Required.</i> Specifies the view for GET to use.	
Format	<i>View-name</i> must be the name of an existing, opened view.	
Options	<i>*</i>	GET with the view you used most recently.
	<i>view-name</i>	GET with the named view.

FOR UPDATE

Description *Optional.* Locks the physical record(s) associated with the row you are retrieving. Prevents others from updating those record(s).

Considerations

- ◆ The FOR UPDATE phrase allows you to perform modifications dependent on the current contents of the view.
- ◆ If you do not need to lock the view, issue a GET without the FOR UPDATE phrase. When you perform the UPDATE or **DELETE** function, the automatic hold facility of RDM performs the lock prior to modifying the row.
- ◆ FOR UPDATE implies that all physical resources remain locked until you issue another GET or an **INSERT**, **UPDATE**, **DELETE**, **COMMIT**, or **RESET**.

AT mark-name

Description *Optional.* Repositions a view previously marked with the **MARK** command.

Consideration You cannot use the USING and AT phrases with the same GET command.

USING *literal*₁ *literal*₂ ...

Description	<i>Optional.</i> Identifies a value or set of values to be used for a keyed GET.
Format	Each literal consists of character, hexadecimal, or numeric data, in one of the following forms:
	'cccccc' Character data
	x'xxxxxx' Hexadecimal data
	'nnnnnnn' Numeric data (with optional quotes)
	nnnnnnn Numeric data (without optional quotes)

Considerations

- ◆ The number of keys specified in the GET statement must be less than or equal to the number of keys in your specified column list. No more than nine keys are allowed in one view.
- ◆ RDM treats any omitted keys as generic keys. The use of generic keys is a convenient feature for allowing both direct access to a view and a sequential scan of many rows. RDM will return all occurrences of a particular unspecified column as long as the other keys are satisfied.
- ◆ The order of specified keys in the USING phrase corresponds to the order of key declarations in your column list.
- ◆ You must specify a valid and non-null value. RDM validates the value before any physical I/O takes place.

Examples

- ◆ This example shows the retrieval of the first row in the CUSTOMER view.

```
> GET CUST
```
- ◆ This example shows a keyed GET.

```
> GET FIRST CUST USING Z95551
```

GO command

The GO command issues a penetration **GET** request followed by a series of sweeping GET requests, and displays the rows in tabular format.

```
GO [ NEXT ] { *
    [ PRIOR ] view - name }

[ START { NEXT
          LAST
          SAME
          FIRST
          PRIOR
          AT mark - name } ]

[ FOR number - of = rows ]

[ { FROM
  { USING } literal1 literal2 ... ]
```

```
[ NEXT
  PRIOR ]
```

- Description** *Optional.* Specifies the **GET** command modifier to be used in retrievals after the initial penetration.
- Default** NEXT
- Consideration** The effect of the positional keywords varies depending on the physical data platform. See the discussion of **positional keywords** at the beginning of this chapter.

```
{ *
  view - name }
```

- Description** *Required.* Specifies the view for GO to use.
- Format** *View-name* must be the name of an existing, opened view.
- Options** * GO with the view you used most recently.
view-name GO with the named view.

START	{	NEXT	}
		LAST	
		START	
		<u>FIRST</u>	
		PRIOR	
		AT <i>mark - name</i>	

Description *Optional.* Specifies the **GET** command modifier to use for the initial penetration of the database.

Default FIRST

Consideration The effect of the positional keywords varies depending on the physical data platform. See the discussion of **positional keywords** at the beginning of this chapter.

FOR *number-of-rows*

Description *Optional.* Indicates the maximum number of rows to be returned.

Format Numeric characters

Consideration GET NEXTs repeat until the count is exhausted or until the last view is retrieved, whichever occurs first.

**{FROM
USING}** *literal₁ literal₂ ...*

Description	<i>Optional.</i> Identifies a value or set of values to be used for a keyed GET .	
Format	Either character or numeric data. You must enclose character data that includes blanks in quotes; numeric data need not be.	
Options	FROM	Use the key values only on the initial penetration; the scan is unqualified.
	USING	Use the key values for both the initial penetration and the subsequent scan.

Considerations

- ◆ The number of keys specified in the **GET** statement must be less than or equal to the number of keys in your specified column list.
- ◆ RDM treats any omitted keys as generic keys. The use of generic keys allows for both direct access to a view and a sequential scan of many rows. RDM returns all occurrences of a particular unspecified key as long as the other keys are satisfied.
- ◆ The order of specified keys in the **USING** phrase corresponds to the order of key declarations in your column list.
- ◆ You must specify a valid and non-null value. RDM validates the value before any physical I/O takes place.

General considerations

- ◆ RDM displays the output in columns. To display more data than will fit on a screen/page, use an alternate format.
- ◆ After the GO command displays a page of rows (see **PAGESIZE**), the prompt ****MORE**** appears. Enter a blank line for each additional page in batch mode; press ENTER for each additional page in online mode.
- ◆ At the end of the series of views retrieved by GO, the prompt ****END**** appears.
- ◆ Do not use “FOR *number-of-rows*” for online because DBAID will not pause until the last screen.
- ◆ The GO command always looks ahead one row so it can determine whether to display the ****MORE**** or ****END**** message. If you issue a **GET** after the GO, a row may appear to have been skipped. To view the row immediately following the last row after the GO, issue a GET SAME.

Examples

- ◆ GO VIEW START AT VIEW-MARK1 USING (key-value) issues the following sequence of RDM **GET** commands:

```
> GET VIEW AT VIEW-MARK1
> GET NEXT VIEW USING (key-value)
> GET NEXT VIEW USING (key-value)
.
.
.
```

until RDM returns a not found FSI.

- ◆ GO PRIOR VIEW START LAST FROM (key-value) issues the following sequence of RDM **GET** commands:

```
> GET LAST VIEW USING (key-value)
> GET PRIOR VIEW
> GET PRIOR VIEW
.
.
.
```

until RDM returns a not found FSI.

INSERT command

The INSERT command places a row in the physical database based on the relative location you specify.

INSERT

NEXT

LAST

FIRST

PRIOR

*

view - name

MASS

NEXT

LAST

FIRST

PRIOR

- Description

Optional. Specifies the relative location of the row you want to insert. The access definition may override this specification.
- Default

NEXT
- Considerations

- ◆

For nonuniquely keyed values:

INSERT FIRST

Places a row in the first position in the view.

INSERT NEXT

Places a row after the current row. If no current position exists, INSERT NEXT places the row in the last position in the view.

INSERT PRIOR

Places a row before the current row. If no current position exists, INSERT PRIOR places the row in the first position in the view.

INSERT LAST

Places a row in the last position of the view.
- ◆

The effect of the positional keywords varies depending on the physical data platform. See the discussion of **positional keywords** at the beginning of this chapter.

**{ *
view - name }**

Description	<i>Required.</i> Specifies the view to use to insert the row(s).
Format	<i>View-name</i> must be the name of an existing, opened view.
Options	<div><div>*</div><div>INSERT with the view you used most recently.</div></div> <div><div><i>view-name</i></div><div>INSERT with the named view.</div></div>

MASS

Description *Optional.* Allows you to insert multiple rows in the physical database.

Considerations

- ◆ Every RDM insert command issued by MASS insert uses the positioning parameter specified (NEXT, LAST, FIRST, or PRIOR).
- ◆ Enter views immediately following this command after the prompt lines MASS INSERT PROCESSING INITIATED and ENTER “END.” TO EXIT MASS INSERT appear.
- ◆ Separate the column values with commas. To insert rows longer than one line, terminate the list of values with a comma and continue the input on the next line.
- ◆ Place multiple rows on a single line by leaving a blank between rows.
- ◆ Use a pair of single quote marks to enclose columns containing spaces.
- ◆ If you have columns with no values, enter two consecutive commas to indicate their absence. RDM treats this as a null value for packed or zoned columns, as a large number (X'40404040' or 67372036 integer) for binary columns, and as blanks for a character column.
- ◆ Specify END. after you enter all rows to be inserted into the view. The period after END is mandatory.

General considerations

- ◆ After you enter the column values on a single insert (not using MASS), RDM displays the view. The message INSERT (Y/N)? appears. Enter a Y response to insert the view. Any other response will not insert the row.
- ◆ Processing stops if RDM detects ten errors while using the MASS insert; otherwise, enter END. to terminate insert processing.
- ◆ After an INSERT, C and V are the only meaningful ASIs.

Examples

The following are examples of using INSERT in an online environment. The > indicates user input.

- ◆ Example of a single insert:

```
> INSERT CUST
CUSTOMER-NO
> A7865
CUSTOMER-NAME
> SSTP
CUSTOMER-ADDR
> 3350 RUTHER
CUSTOMER-CITY
> CINCINNATI
CUSTOMER-STATE
> OH
CUSTOMER-ZIPCODE
> 45220
CUSTOMER-CLASS
> T8
CUSTOMER-CR-LIM
> 750.00
CUSTOMER-BRANCH
> 1261
CUSTOMER-NO                ( ) A7865
CUSTOMER-NAME              ( ) SSTP
CUSTOMER-ADDR              ( ) 3350 RUTHER
CUSTOMER-CITY              ( ) CINCINNATI
CUSTOMER-STATE             ( ) OH
CUSTOMER-ZIPCODE           ( ) 45220
CUSTOMER-CLASS             ( ) T8
CUSTOMER-CR-LIM            ( ) 750.00
CUSTOMER-BRANCH            ( ) 1261
INSERT (Y/N)?
> Y
FSI: * VSI: + MSG: SUCCESSFUL COMPLETION
```

◆ Example of a MASS insert (first row):

```
> INSERT * MASS
MASS INSERT PROCESSING INITIATED.
ENTER "END." TO EXIT MASS INSERT.
> 9997,BBBB,100783
FSI: * VSI + MSG: SUCCESSFUL COMPLETION
```

Example of same MASS insert (using comma to continue to next line):

```
> 9996,CCCC,
> 100683
FSI * VSI: + MSG: SUCCESSFUL COMPLETION
```

Example of same MASS insert (multiple rows on a single line):

```
> 9995,DDDD,100583 9994,EEEE,100483 9993,FFFF,100383
FSI * VSI: + MSG: SUCCESSFUL COMPLETION
FSI * VSI: + MSG: SUCCESSFUL COMPLETION
FSI * VSI: + MSG: SUCCESSFUL COMPLETION
```

Ending the MASS insert processing:

```
> END.
MASS INSERT PROCESSING COMPLETED.
```

KEEP command

The KEEP command disables the DBAID automatic **RESET** facility. This command is the opposite of the **ERASE** command. This command prohibits DBAID from issuing a RESET when it receives an X FSI from the view. Instead, DBAID keeps the database as it is and lets the user decide whether to RESET. This is the default setting.

KEEP

General considerations

- ◆ KEEP is the default.
- ◆ KEEP does not affect the **RESET** that systems commands may issue (**REMOVE**, **SAVE**, **BIND**, **PERMIT**, and **DENY**) when an error occurs.

Line-number command

The line-number command deletes, adds, or replaces a view definition statement in the virtual view currently being edited or defined.

***line-number* [*view-definition-statement*]**

line-number

Description *Required.* Indicates the number of the line to delete, add, or replace.

Format 1–4 numeric characters

Considerations

- ◆ If a line number is less than four digits, DBAID adds zeroes to the front of the number. For example, 10 becomes 0010. If the number is longer than four digits, DBAID truncates it to the first four digits.
- ◆ If you use the line-number command without a following ddl-statement line, this command deletes the line from the view definition.

view-definition-statement

Description *Optional.* Specifies the view definition statement to add or replace.

Format Must be a valid view definition statement.

General considerations

- ◆ Before you can use this command, you must first have issued a **DEFINE**, **EDIT**, or **LIST** command.
- ◆ Entering a command other than line-number terminates the **DEFINE**, **EDIT**, or **LIST** command.

Example

This example first lists the view, and when you type in the line number, modifies the lines.

```
> LIST CUST
0100 CUSTOMER-NO
0200 CUSTOMER-ADDR
0300 CUSTOMER-CITY
0400 ACCESS CUST
> 100 KEY CUSTOMER-NO          *Replaces line 100
> 150 CUSTOMER-NAME           *Inserts line 150
> 300                          *Deletes line 300
```

DBAID lists the modified view again.

```
> LIST CUST
0100 KEY CUSTOMER-NO
0150 CUSTOMER-NAME
0200 CUSTOMER-ADDR
0400 ACCESS CUST
```

LINESIZE command

The LINESIZE command specifies the number of characters to display in a line.

LINESIZE [*number-of-characters*]

number-of-characters

Description *Optional.* Indicates the number of characters to display on a line.

Default 79

Format 2–3 numeric characters.

Options 10–132

Considerations

- ◆ In an online environment, the screen size restricts the line size maximum to the line capacity of the screen.
- ◆ If you omit the number-of-characters, the command displays the current LINESIZE setting.

LIST command

The LIST command displays a saved or virtual view and readies it for modification.

```
LIST [schema-name:] { *
                      view - name }
```

schema-name:

- Description** *Optional.* Identifies the schema in which the view to be listed is located.
- Format** Must be a valid schema defined on the Directory. The schema name must be followed by a colon.
- Consideration** If you omit this parameter, LIST uses the active schema.

```
{ *
  view - name }
```

- Description** *Required.* Specifies the view to list.
- Format** *View-name* must be the name of an existing view.
- Options** * List the view you used most recently.
- view-name* List the named view.
- Consideration** If the *view-name* is not a virtual view, DBAID searches for the view in the Directory.

General considerations

- ◆ Once you issue the LIST command, you can use the *line-number* command to modify your view.
- ◆ If a LIST command returns no definition for a view, you may have opened the view without doing a LIST or **DEFINE** of the view in the current session. To remedy this, do a **RELEASE**, then **UNDEFINE**, and then LIST. You must open the view again to execute it.
- ◆ LIST automatically issues an **EDIT** command for this view.
- ◆ Using LIST prior to **OPEN** reads the text for the view definition from the Directory. A subsequent open of this view will not perform any security checking on whether the view is related to the user.
- ◆ DBAID can create views when you enter text with LIST, **DEFINE**, or **EDIT**. If you use LIST on a view in the Directory, the text becomes a virtual view and DBAID can modify it. Virtual views let you open a view without relating it to a user. A **SAVE** command does not relate the DBAID user to the view.

Examples

- ◆ The following lists the BRANCH-VIEW from the active schema:

```
> LIST BRANCH-VIEW
0100 KEY      BRANCH-NO
0200          BRANCH-NAME
0300          BRANCH-ADDR
0400          BRANCH-CITY
0500          BRANCH-STATE
0600          BRANCH-ZIPCODE
0700 REQ      BRANCH-REGION = BRANCH-REGION = REGION-NO
0800 ACCESS E$BR WHERE BRANCH-NO = BRANCH-NO ALLOW ALL
0900 ACCESS E$RG ONCE WHERE REGION-NO = BRANCH-REGION
1000 ACCESS E$CU WHERE CUSTOMER-BRANCH = BRANCH-NO
```

- ◆ The following lists the view VIEW from the Directory. OTHERSCH is not the active schema.

```
> LIST OTHERSCH: VIEW
```

MARK command

The MARK command marks the current position of the view that the previous **GET** command established.

MARK {^{*}
view - name } **AT** *mark-name*

{^{*}
view - name }

Description	<i>Required.</i> Specifies the view for which the position is to marked.
Format	<i>View-name</i> must be the name of an existing, opened view.
Options	<div><div>[*]</div><div>Mark the position for the view you used most recently.</div></div> <div><div><i>view-name</i></div><div>Mark the position for the named view.</div></div>

AT *mark-name*

Description	<i>Required.</i> Assigns a name to the location where you want to mark the position of the current view.
Format	1–30 alphanumeric characters and the special characters #, \$ and -. The first character must be alphabetic or a special character. If the first character is a special character, the second character must be alphabetic.
Consideration	The name assigned is the name you will use in a later GET AT request to retrieve using this view.

General considerations

- ◆ Use the AT clause in the **GET** command to reposition the view at the position set by the MARK command and named by mark-name.
- ◆ You can create any number of MARKs for a view, but to conserve internal memory, it is best to reuse the mark-name when possible.
- ◆ The number of MARKs you can create is limited by the amount of internal memory space allocated to your task.
- ◆ The size of the available slot limits the number of MARKs a program can have outstanding at any time. When the program no longer requires a particular MARK, issue a **FORGET** command for the data-item.

MARKS command

The MARKS command lists all open **MARKS** and the views they are marking.

MARKS

Example output

```
> MARKS
```

	MARK NAME	VIEW NAME
	B33593	CUST
	Z9551	CUST
	H2233	CUST
	CASH	CUST

OPEN command

The OPEN command readies a stored or virtual view for use by DBAID.

OPEN [user-view-name=]

*

view - name

[column₁ column₂...]

user-view-name=

Description

Optional. Gives an existing view a name to be used in DBAID.

Format

1–30 alphanumeric characters and the special characters #, -, and \$. The first character must be alphabetic or a special character. If the first character is a special character, the second character must be alphabetic.

Considerations

- ◆ If you do not specify *user-view-name*, it will be the same name as the *view-name*.
- ◆ You can use this method (together with the column parameter) to create many smaller views from one common view.
- ◆ To OPEN a view that has not been listed or defined in the same session of DBAID, the user must be related to the view in the Directory.

*

view - name

Description

Required. Specifies the view to be readied for use.

Format

View-name must be the name of an existing view.

Options

*

Open the view you used most recently.

view-name

Open the named view.

Consideration

If there is a virtual view with the same name as a saved view, DBAID uses the virtual view.

column₁ column₂ ...

Description *Optional.* Identifies the column or list of columns to include in the user view. If omitted, all columns in the view are in the user view.

Format The columns must be specified in the view being opened.

Consideration The list of column names may be continued on successive lines by ending the line you are entering with a comma. The command **USER-LIST** displays the list of columns used to open the view after it has been opened.

General considerations

- ◆ The OPEN command returns a message showing the number of bytes of memory used by the view:

```
nnnnn BYTES USED IN OPENING VIEW
```

This information can help determine run-time internal memory requirements.

- ◆ Issuing an OPEN request on a view without first issuing a **LIST** request opens the view with the user relations checked but without the view definition text available to DBAID.
- ◆ If changes are made to a view, you must issue the **RELEASE** command and OPEN the view to implement the changes.

Example This example will return only BRANCH-NO and BRANCH-NAME when you do a **GET**, even though BRANCH-VIEW has more columns defined.

```
> OPEN BRANCH-NO-AND-NAME = BRANCH-VIEW BRANCH-NO, BRANCH-NAME
```

PAGESIZE command

The PAGESIZE command specifies the maximum number of lines to display on a screen (in online DBAID) or on a page (in batch DBAID).

PAGESIZE [nnn]

number-of-lines

Description *Optional.* Specifies the maximum number of lines to display on a screen (in online DBAID) or on a page (in batch DBAID).

Default 24

Format 2 or more numeric characters

Considerations

- ◆ The PAGESIZE number must be greater than 10.
- ◆ In an online environment, the PAGESIZE number cannot exceed the screen capacity.
- ◆ If you omit the number from the PAGESIZE command, RDM displays the current PAGESIZE number.

PERMIT command

The PERMIT command permits specified users to use a view by relating the view to the user(s) in the Directory.

PERMIT {^{*}
view - name } *user-name*₁ [{^b
, } *user - name*_{2...}]

{^{*}
view - name }

Description *Required.* Specifies the view to be related to a user.

Format *View-name* must be the name of an existing view.

Options * Relate the view you used most recently.

 view-name Relate the named view.

*user-name*₁ [{^b
, } *user - name*_{2...}]

Description *Required.* The name(s) of the user(s) you are relating to the view in the Directory. User names must be separated by a comma or a blank.

Format Each *user-name* must be the name of an existing user defined on the Directory.

Consideration You can make this view available to all users by specifying the ****PUBLIC**** user.

General considerations

- ◆ You can use the DBAID PERMIT command instead of using the Directory Maintenance RELATE function.
- ◆ After successfully relating the view to each user, DBAID commits the Directory update.
- ◆ If an error occurs while relating a user, DBAID backs out (resets) the Directory update and terminates processing of the command.
- ◆ You can use this command only if your active environment description specifies update access to the Directory.

PRINT-STATS command

The PRINT-STATS command causes RDM to print the current statistics.

PRINT-STATS

General considerations

- ◆ The **STATS-ON** command must precede the first PRINT-STATS command. If you do not first issue STATS-ON, PRINT-STATS has no effect.
- ◆ Issue a **STATS-OFF** to stop gathering statistics.
- ◆ RDM routes statistics output to the DMLPRINT output file.
- ◆ Use the PRINT-STATS command to keep a statistical running total.

Example

In the following example, PRINT-STATS prints statistics after each RDML operation:

```
> STATS-ON
> GET NEXT BRANCH-VIEW
.
.
> PRINT-STATS
> UPDATE BRANCH-VIEW
.
.
> PRINT-STATS
```


PUBLIC-DENY command

The PUBLIC-DENY command removes the relationship between the specified views and the ****PUBLIC**** user on the Directory. A view not related to the ****PUBLIC**** user can be used only by a user explicitly related to that view.

```
PUBLIC-DENY view-name1  $\left[ \left\{ \begin{array}{c} \mathbf{b} \\ , \end{array} \right\} \right] \mathbf{view} - \mathbf{name2...}$ 
```

```
view-name1  $\left[ \left\{ \begin{array}{c} \mathbf{b} \\ , \end{array} \right\} \right] \mathbf{view} - \mathbf{name2...}$ 
```

Description *Required.* Specifies the name(s) of the view(s) for which you want to revoke public access.

Format Must be the name(s) of valid view(s) defined on the Directory. View names must be separated by a comma or a blank.

General considerations

- ◆ The ****PUBLIC**** user must be defined on the Directory.
- ◆ You can use the DBAID PUBLIC-DENY command instead of using Directory Maintenance to delete the relationship between the view and the ****PUBLIC**** user.
- ◆ You can use the PUBLIC-DENY command to remove the relationship between the ****PUBLIC**** user and the view regardless of whether you created the relationship with Directory Maintenance or the **PUBLIC-PERMIT** command.
- ◆ After successfully removing the relationship between each view and the ****PUBLIC**** user, DBAID commits the Directory update.
- ◆ If an error occurs while removing the relationship, DBAID backs out (resets) the Directory update and terminates processing of the command.
- ◆ You can use this command only if your active environment description specifies update access to the Directory.
- ◆ Global views that are related to the ****PUBLIC**** user continue to be accessible to all RDM users until RDM is reinitialized.

PUBLIC-PERMIT command

The PUBLIC-PERMIT command relates view(s) to the ****PUBLIC**** user on the Directory. All RDM users can use views related to the ****PUBLIC**** user.

PUBLIC-PERMIT *view-name*₁ $\left[\left\{ \begin{array}{c} \text{b} \\ , \end{array} \right\} \right]$ *view - name*_{2...}

*view-name*₁ $\left[\left\{ \begin{array}{c} \text{b} \\ , \end{array} \right\} \right]$ *view - name*_{2...}

Description *Required.* Specifies the name(s) of the view(s) for which you want to grant public access.

Format Each *view-name* must be the name of an existing view defined on the Directory. View names must be separated by a comma or a blank.

General considerations

- ◆ The ****PUBLIC**** user must be defined on the Directory.
- ◆ You can use the DBAID PUBLIC-PERMIT command instead of using the Directory Maintenance RELATE function to relate views to the ****PUBLIC**** user.
- ◆ After successfully relating each of the views to the ****PUBLIC**** user, DBAID commits the Directory update.
- ◆ If an error occurs while removing the relationship, DBAID backs out (resets) the Directory update and terminates processing of the command.
- ◆ You can use this command only if your active environment description specifies update access to the Directory.
- ◆ Making the public views global improves performance.

PUBLIC-VIEWS command

The PUBLIC-VIEWS command lists the names and short text for the views related to the ****PUBLIC**** user.

PUBLIC-VIEWS

Consideration The format of the display is identical to the display produced by the **VIEWS-FOR-USER** DBAID command.

RELEASE command

The RELEASE command closes specified view(s) and releases the RDM memory they occupy.

```
RELEASE [ *  
        view - name ]
```

```
[ *  
view - name ]
```

- Description** *Optional.* Specifies the view to release.
- Format** *View-name* must be the name of an existing, opened view.
- Options** * Release the view you used most recently.
- view-name* Release the named view.
- Consideration** If you omit this parameter, the RELEASE command releases all of your opened views.

General consideration

This command does not affect virtual view text of the view(s).

REMOVE command

The REMOVE command removes the view definition text, its binding if the view is bound, and the relation between the view and users, procedures, external columns, and environment descriptions.

REMOVE [*schema-name*:] {
*
view - name}

schema-name:

- Description

Optional. Identifies the schema where the view to be removed is defined.
- Format

Must be the name of an existing schema defined on the Directory. The schema name must be followed by a colon.
- Consideration

If you omit this parameter, REMOVE uses the active schema.

{
*
view - name}

- Description

Required. Specifies the view to remove.
- Format

View-name must be the name of an existing view that you have listed or edited.
- Options

*
view-name

Remove the view you used most recently.
Remove the named view.

General considerations

- ◆ You must list the view before you can remove it. This protects you from inadvertently removing views due to spelling errors.
- ◆ DBAID automatically issues a **COMMIT** when the REMOVE command completes successfully.
- ◆ If an error occurs while modifying the Directory, DBAID automatically issues a **RESET** and processing stops.
- ◆ You can use this command only if your active environment description specifies update access to the Directory.

Examples

- ◆ The following removes all view definition text, binding, and relations to the view PROD-VIEW in the Directory for the active schema. The view is still a virtual view in DBAID.

```
> REMOVE PROD-VIEW
```

- ◆ The following removes all view definition text, binding, and relations to the view PROD-VIEW in the Directory for the schema OTHERSCH. The active schema is unaffected.

```
> REMOVE OTHERSCH: PROD-VIEW
```

RENUMBER command

The RENUMBER command renumbers a virtual view so that the line numbering starts at ten, with each line incremented by ten.

RENUMBER {^{*}
[*view - name*]}

{^{*}
[*view - name*]}

Description	<i>Required.</i> Specifies the view to renumber.
Format	<i>View-name</i> must be the name of an existing, opened view.
Options	<div><div>*</div><div>Renumber the view you used most recently.</div></div> <div><div><i>view-name</i></div><div>Renumber the named view.</div></div>

RESET command

The RESET command rolls back any database updates since the last **COMMIT** point.

RESET

General considerations

- ◆ Use RESET only after unsuccessful RDML updates. The DBAID default is to not automatically issue a RESET command when RDM returns an X FSI. See the **KEEP** and **ERASE** commands.
- ◆ The system commands **REMOVE**, **SAVE**, **BIND**, **PERMIT**, and **DENY** issue a reset if an error occurs while modifying the Directory.
- ◆ In batch mode with TLR, a RESET backs out any database updates since the last **COMMIT**. It does not restart DBAID.
- ◆ In batch mode without TLR, a RESET causes an intentionalabend.

SAVE command

The SAVE command stores views in the Directory for either the active schema or a specified schema. This command allows you to bind views to improve open performance.

```
SAVE [schema-name:] { *
                      view - name } [BIND]
```

schema-name:

Description *Optional.* Identifies the schema in which the view to be saved is located.

Format Must be a valid schema defined on the Directory. The schema name must be followed by a colon.

Consideration If you omit this parameter, SAVE uses the active schema.

```
{ *
  view - name }
```

Description *Required.* Specifies the view to stored on the Directory.

Format *View-name* must be the name of an existing virtual view.

Options * Save the view you used most recently.

view-name Save the named view.

Consideration The view must have been created using one of the editing commands: **DEFINE**, **EDIT**, or **LIST**.

BIND

Description *Optional.* Indicates that you want to bind the view.

Considerations

- ◆ RDM stores bindings on the Directory under the View entity in the Directory. You can use all Directory Maintenance functions, except **LIST**, on the bound view.
- ◆ See “**View binding**” on page 125 for more information on binding.

General considerations

- ◆ If the view you are saving already exists, the system asks if you want to replace the existing view. If yes, the new view replaces the old view on the Directory. If the view did not previously exist, you must relate it to users before application programs can access it.
- ◆ You can use this command only if your active environment description specifies update access to the Directory.
- ◆ When the SAVE completes successfully, DBAID automatically issues a **COMMIT**. If you use the BIND option, DBAID issues a COMMIT at the completion of the SAVE and the completion of the BIND.
- ◆ If an error occurs while modifying the Directory, DBAID automatically issues a **RESET** and processing stops.

Examples

- ◆ The following stores BRANCH-VIEW's view definition under the active schema:

```
> SAVE BRANCH-VIEW
```
- ◆ The following stores BRANCH-VIEW's view definition under the active schema and binds the view under the active schema:

```
> SAVE BRANCH-VIEW BIND
```
- ◆ The following stores BRANCH-VIEW's view definition under the schema OTHERSC:

```
> SAVE OTHERSC: BRANCH-VIEW
```

SHOW-NAVIGATION command

The SHOW-NAVIGATION command displays the access strategies RDM uses for entities (files/views) accessed in open views.

```
SHOW-NAVIGATION [*  
                  view - name]
```

```
[*  
  view - name]
```

- | | | | | | |
|----------------------|--|----------|--|------------------|---|
| Description | <i>Optional.</i> Specifies the view for which you wish to display access strategies. | | | | |
| Format | <i>View-name</i> must be the name of an existing, opened view. | | | | |
| Options | <table><tr><td><i>*</i></td><td>Display access strategies for the view you used most recently.</td></tr><tr><td><i>view-name</i></td><td>Display access strategies for the named view.</td></tr></table> | <i>*</i> | Display access strategies for the view you used most recently. | <i>view-name</i> | Display access strategies for the named view. |
| <i>*</i> | Display access strategies for the view you used most recently. | | | | |
| <i>view-name</i> | Display access strategies for the named view. | | | | |
| Consideration | You must first have used one of the following commands on the view: DEFINE , EDIT , or LIST . | | | | |

Example The following example shows the definition of a view, followed by a SHOW-NAVIGATION command and the output of the command:

```
> DEFINE BRANCH-BASE-VIEW
> 0100 KEY      BRANCH-NO
> 0200          BRANCH-NAME
> 0300          BRANCH-ADDR
> 0400          BRANCH-CITY
> 0500          BRANCH-STATE
> 0600          BRANCH-ZIPCODE
> 0700          BRANCH-DEL-ROUTE
> 0800          BRANCH-SLS-QUOTA
> 0900          BRANCH-STF-QUOTA
> 1000 REQ      BRANCH-REGION = BRANCH-REGION = REGION-NO
> 1100 ACCESS E$BR WHERE BRANCH-NO = BRANCH-NO ALLOW ALL
> 1200 * REJECT INSERT AND UPDATE OF BRANCH- REGION IF REGION NOT VALID
> 1300 ACCESS E$RG ONCE WHERE REGION-NO = BRANCH-REGION
> 1400 * REJECT DELETION OF A BRANCH THAT HAS CUSTOMERS
> 1500 ACCESS E$CU WHERE CUSTOMER-BRANCH = BRANCH-NO
> OPEN *
FSI: *   VSI: =   MSG:    4536 BYTES USED IN OPENING VIEW.
> SHOW-NAVIGATION
```

Example output

VIEW-NAME	:	BRANCH-BASE-VIEW			
LVL	ACCESSED	FILE/VIEW	NAME	ACCESS METHOD	ACCESS PATH NAME
0	E\$BR			KEYED	E\$BRCTRL
0	E\$RG			KEYED	E\$RGCTRL
1	E\$CU			INDEXED	E\$CUSK01

SIGN-OFF command

The SIGN-OFF command signs off the user from RDM.

SIGN-OFF

General considerations

- ◆ Use the SIGN-OFF command to sign off as an RDM user without terminating the DBAID session.
- ◆ When you terminate the DBAID session with the **BYE** command, RDM signs you off automatically. You need not issue a SIGN-OFF command before BYE.
- ◆ The SIGN-OFF command turns off statistics.

SIGN-ON command

The SIGN-ON command identifies the user to RDM.

SIGN-ON *user-name* [:*psbname*] [*password*]

user-name

Description	<i>Required.</i> Specifies the user.
Format	Must be the name of an existing user defined on the Directory.

psbname

Restriction	For use with IMS databases only.
Description	<i>Optional.</i> Specifies the name of the Program Specification Block (PSB) to be used to access IMS databases.
Format	Must be the name of a PSB in the PSB library.

password

Description	<i>Optional.</i> Specifies the user's password.
Format	Must be the password defined for the user on the Directory.

General considerations

- ◆ In an online environment, you can issue SIGN-ON before entering DBAID and you need not repeat it.
- ◆ In batch mode, the password field does not print on the output.

Example The following shows the user, JDOE, signing on with the password, "DBAPSWD:"

```
> SIGN-ON JDOE DBAPSWD
```

STATS command

The STATS command causes RDM to display online the current statistics of all open views or on the view you specify. You can issue the STATS command numerous times during a session after you have issued a **STATS-ON** command.

```

STATS [ *
      [ view - name ]

```

```

[ *
  [ view - name ]

```

Description	<i>Optional.</i> Specifies the view for which you wish to display statistics.
Format	<i>View-name</i> must be the name of an existing, opened view.
Options	<div>* Display statistics for the view you used most recently.</div> <div><i>view-name</i> Display statistics for the named view.</div>

General considerations

- ◆ The **STATS-ON** command must precede the first STATS command; if you do not issue STATS-ON first, STATS has no effect.
- ◆ You can issue a **STATS-OFF** to stop gathering statistics.
- ◆ STATS displays statistics on your online terminal.
- ◆ You can use the STATS command to keep a running total.

Example In the following example, STATS is used to display statistics after each RDML operation:

```

> STATS-ON
> GET NEXT CUST
.
.
> STATS
> UPDATE CUST
.
.
> STATS

```

STATS-OFF command

The STATS-OFF command causes RDM to print the current statistics, and disables statistics gathering after printing.

STATS-OFF

General considerations

- ◆ The **STATS-ON** command must precede the STATS-OFF command.
- ◆ RDM routes statistics output to the DMLPRINT output file.
- ◆ Issuing the STATS-OFF command without a preceding **STATS-ON** command has no effect.
- ◆ The **BYE** or **SIGN-OFF** commands turn statistics off without printing them.

STATS-ON command

The STATS-ON command causes RDM to initialize the statistics to zero and then begin gathering statistics on a user area. The DBA can use this command in conjunction with the **STATS-OFF**, **PRINT-STATS**, or **STATS** commands to examine what user views do on both a logical and physical level.

STATS-ON

General considerations

- ◆ RDM gathers statistics on a task basis, not on a systemwide basis.
- ◆ Use the **STATS-OFF** command to print statistics and then turn them off.
- ◆ Use the **PRINT-STATS** command to print statistics and continue gathering a running total.
- ◆ Use the **STATS** command to display statistics online and continue gathering a running total.
- ◆ If you issue the **BYE** or **SIGN-OFF** command, RDM turns off your statistics without printing them.

SURE command

The SURE command causes a COMMIT after each successful insert, update, or delete. The SURE command causes RDM to automatically issue a COMMIT if an RDML command returns an “*” FSI that alters the database. This is the opposite of the CAUTIOUS command. This is the default setting.

SURE

UNDEFINE command

The UNDEFINE command removes a virtual view.

UNDEFINE { *
ALL
view - name }

{ *
ALL
view - name }

Description	Required. Specifies which virtual view(s) to remove.	
Options	*	Remove the virtual view whose name you used most recently
	ALL	Remove all virtual views
	view-name	Remove the virtual view named.

General considerations

- ◆ The UNDEFINE command releases the memory used by the view, allowing it to be reclaimed for defining other views. If storage is not relinquished and no more space is available, DBAID issues a 2816 abend code the next time you issue an OPEN or LIST command.
- ◆ This command has no effect on a view definition saved on the Directory.
- ◆ If you want to save a view and release its memory, issue SAVE before UNDEFINE.
- ◆ If the view is currently open, the UNDEFINE command does not “release” the space used by RDM for the view.

UPDATE command

The UPDATE command updates data values in the database.

```
UPDATE { *
        view - name } [column1:=literal1,column2:=literal2 ...]
```

```
{ *
  view - name }
```

Description	Required. Specifies the view you want to update.	
Format	View-name must be the name of an existing, opened view.	
Options	*	Update the view you used most recently.
	view-name	Update the named view.

column₁:=literal₁,column₂:=literal₂ ...

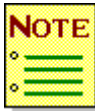
Description *Optional.* Identifies column(s) in the view and their intended values.

Format

<i>columnn</i>	The name of an existing column in the view
<i>literal</i>	Character data or numeric data

Considerations

- ◆ If you do not specify column name(s) in the UPDATE command, DBAID displays the name of each column. Each time it displays the name of an updateable column, it prompts you for a replacement value. After processing all columns, DBAID displays the prompt UPDATE (Y/N) and requires a response.
- ◆ When DBAID prompts you for a replacement value in an online environment, and you just press ENTER, the column's original value is unchanged. When DBAID prompts you for a replacement value in batch mode, the contents of your next input record become the new value, even if it is all blanks.



In batch mode, Cincom recommends you specify column names in the UPDATE command.

- ◆ If you specify column names in the UPDATE command, only the values of the columns you specify are updated. All others remain the same.
- ◆ Do not use single quotes around numeric literals.
- ◆ Single quotes are optional around character literals that contain alphanumeric characters only (no spaces or special characters).
- ◆ In online DBAID only, you must use single quotes to change the value of a column to blanks. A literal of spaces (keyed in) must be in single quotes. If you just press ENTER, you do not affect the column's value.
- ◆ You cannot use the UPDATE function to modify key column values.
- ◆ To UPDATE a row, you must first retrieve the row with the **GET** command.
- ◆ UPDATE can change only one row at a time. For example, to change all PROD-CODES to T100, you must **GET** and UPDATE each row individually.

General consideration

After an UPDATE, C and V are the only meaningful ASIs.

Example

This example shows two columns in the CUST-PROD view being updated.

```
> UPDATE BRANCH BRANCH-NAME = OAKLEY, BRANCH-REGION = 333
```

USER-LIST command

The USER-LIST command displays the column list for the user view named.

USER-LIST {^{*}
view - name}

{^{*}
view - name}

- | | |
|--------------------|--|
| Description | <i>Required.</i> Specifies the view whose columns you want to list. |
| Format | <i>View-name</i> must be the name of an existing, opened view. |
| Options | <div><div>[*]</div><div>List the columns for the view you used most recently.</div></div> <div><div><i>view-name</i></div><div>List the columns for the named view.</div></div> |
| Example | This example shows a list of all the columns in the CUSTOMER user view. |

```
> USER-LIST CUST
USER VIEW NAME :    CUSTOMER
VIEW NAME : CUST
USER VIEW LIST :
CUST-NO , CUSTOMER-NAME , CUSTOMER-ADDR , END .
```

VIEW-DEFN command

The VIEW-DEFN command displays a condensed description of a view.

```
VIEW-DEFN [ *
            view - name ]
```

```
[ *
  view - name ]
```

- Description

Optional. Specifies the view whose condensed description you want to display.
- Format

View-name must be the name of an existing, opened view.
- Options

*Display description of the view you used most recently.

view-nameDisplay description of the named view.
- Consideration

If you omit a view-name, RDM displays a condensed description of all your open views.

Example

```
> VIEW-DEFN
VIEW-NAME          ( + ) CUSTOMER
INS-ORDER           ( + ) N
TOTAL-SIZE          ( + ) 93
TOTAL-FIELDS        ( + ) 10
TOTAL-LEVELS        ( + ) 2
TOTAL-DELETABLE     ( + ) 10
TOTAL-INSERTABLE    ( + ) 10
TOTAL-REPLACEABLE   ( + ) 10
TOTAL-REQUIRED      ( + ) 2
TOTAL-KEYS          ( + ) 2
TOTAL-NONUNIQUE     ( + ) 0
***MORE***
```

VIEWS command

The VIEWS command displays all views currently active in DBAID.

VIEWS

General consideration

The information displayed with this command includes:

- ◆ User View Name of the user view.
- ◆ View Name of the view of which this user view is part.
- ◆ Status Indicates whether the user view is open or released.

Example

```
> VIEWS
USER VIEW          VIEW          STATUS
CUS                CUST          OPENED
BRAN               BRAN          OPENED
BRANCH-VIEW       BRANCH-NO     RELEASED
INVOICE           INVOICE       OPENED
```

VIEWS-FOR-USER command

The VIEWS-FOR-USER command lists the names and short text for each view related to the signed-on user or to the ****PUBLIC**** user.

VIEWS-FOR-USER

Consideration The VIEWS-FOR-USER command does not list any view more than once, even if that view is related both to the signed-on user and to the ****PUBLIC**** user.

Example

```

> VIEWS-FOR-USERS
      VIEW NAME                                DATE          TIME
      SHORT DESCRIPTION
MANIFEST                                01/21/91    16:45:43
VIEW OF MANIFEST INFORMATION
REGN                                01/21/91    16:43:55
BRAN                                01/21/91    16:43:46
MANF                                01/21/91    16:43:26
CUST                                01/21/91    16:43:10
INVC                                01/21/91    16:42:58
.....
**MORE**

```

8

Using the RDM reports

The RDM report utility generates reports about RDM views in the active schema. (To track a user/view relationship across multiple schemas, you must use Directory reports. For information about Directory reports, refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250.)

You specify three parameters for a report: Report type, View, and User.

You specify the RDM report(s) you want with the following report type codes:

Code	Meaning
A	All Reports
C	COBOL Programmer's Report
D	DBAs Report
E	Impact of Change (Extract)
L	PL/1 Programmer's Report
P	Both COBOL and PL/1 Programmer's Reports
U	End user Report
V	Views Used by Programs Report

If you specify USER=ALL and a single view, the printed report shows all users for a view. If you specify VIEW=ALL and a single user, the report shows all the views a particular user is able to access. If you specify a single view and a single user, the report utility verifies that they are related.

Your SUPRA Server libraries contain procedures and job control language (JCL) samples for running RDM reports. Samples are subject to change. See the SUPRA Server JCL library or source statement library member TXJ\$INDEX for a list of JCL samples.

OS/390

See the SUPRA Server procedure library member TIS\$RDM for a list of RDM procedures. See the SUPRA Server macro library or source statement library member TX\$\$INDEX for an index to the different kinds of samples. For more information on JCL samples, refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250.

DBA report

The DBA Report helps you keep track of the views you have defined in the Directory and also the users of those views. The DBA Report consists of the following:

- ◆ **TITLE.** Report title includes the date and time the report was generated and the schema name which the report includes.
- ◆ **VIEW.** The name of the view being described.
- ◆ **LAST UPDATE.** The time and date of the last update to this view.
- ◆ **THIS VIEW IS NOT BOUND.** A message indicating the view is not bound.
- ◆ **THIS VIEW IS BOUND.** A message indicating the view is bound.
- ◆ **ACCESS SET.** The view definition, including column and access definitions, as defined on the Directory.
- ◆ **COLUMN.** Lists each column in the view.
- ◆ **FROM.** If reporting on a base view, EXT FIELD = lists the external field this column maps to. If reporting on a derived view, COLUMN = lists the column that this column maps to in another view.
- ◆ **IN.** If reporting on a base view, PHY FIELD = displays the physical field name, and FILE = lists the file name. RC = lists the record code, if applicable. If reporting on a derived view, VIEW = lists the view name the view accesses.
- ◆ **USERS.** Lists the users to whom this view is related.

The following code listing shows a sample DBA Report.

```
SUPRA RDM DIRECTORY REPORTS  LEVEL nnnn COPYRIGHT 19nn CSI - ALL RIGHTS RESERVED 14:46:39 04-01-1991 PAGE 2
*** RELATIONAL DATA MANAGER DBA REPORT FOR SCHEMA BURRYSCH ***
SUPRA RDM DIRECTORY REPORTS LEVEL nnnn COPYRIGHT 19nn CSI - ALL RIGHTS RESERVED 14:46:39 04-01-1991 PAGE 160
*** RELATIONAL DATA MANAGER DBA REPORT FOR SCHEMA BURRYSCH ***

LOGICAL VIEW: SUPPLIERS-BY-PRODUCT
LAST UPDATE : 08:15:50 03-25-1991
THIS LOGICAL VIEW IS NOT BOUND.
ACCESS SET:
100 KEY PRODUCT-CODE
200 PRODUCT-DESC
300 PRODUCT-PRICE
400 PRODUCT-WH-QNTY
500 VS-NO-SUPPLIER
600 VS-NO-PART-NO
700 VS-NO-PART-COST
800 SUPPLIER-NAME
900 SUPPLIER-ADDR
1000 SUPPLIER-CITY
1100 SUPPLIER-STATE
1200 SUPPLIER-ZIPCODE
1300 ACCESS E$PD WHERE PRODUCT-CODE = PRODUCT-CODE
1400 ACCESS E$VS WHERE VS-NO-PRODUCT = PRODUCT-CODE
1500 ACCESS E$SU WHERE SUPPLIER-NO = VS-NO-SUPPLIER

COLUMN                                FROM                                IN

PRODUCT-CODE                         EXT FIELD = PRODUCT-CODE          PHY FIELD = E$PDCTRL FILE = E$PD RC =
PRODUCT-DESC                         EXT FIELD = PRODUCT-DESC          PHY FIELD = E$PDDESC FILE = E$PD RC =
PRODUCT-PRICE                       EXT FIELD = PRODUCT-PRICE          PHY FIELD = E$PDPRCE FILE = E$PD RC =
PRODUCT-WH-QNTY                     EXT FIELD = PRODUCT-WH-QNTY        PHY FIELD = E$PDWQTY FILE = E$PD RC =
VS-NO-SUPPLIER                      EXT FIELD = VS-NO-SUPPLIER          PHY FIELD = E$VSE$SU FILE = E$VS RC =
VS-NO-PART-NO                      EXT FIELD = VS-NO-PART-NO          PHY FIELD = E$VSNUMB FILE = E$VS RC =
VS-NO-PART-COST                     EXT FIELD = VS-NO-PART-COST        PHY FIELD = E$VSV CST FILE = E$VS RC =
SUPPLIER-NAME                       EXT FIELD = SUPPLIER-NAME          PHY FIELD = E$SUNAME FILE = E$SU RC =
SUPPLIER-ADDR                       EXT FIELD = SUPPLIER-ADDR          PHY FIELD = E$SUADDR FILE = E$SU RC =
SUPPLIER-CITY                       EXT FIELD = SUPPLIER-CITY          PHY FIELD = E$SUCITY FILE = E$SU RC =
SUPPLIER-STATE                      EXT FIELD = SUPPLIER-STATE          PHY FIELD = E$SUSTAT FILE = E$SU RC =
SUPPLIER-ZIPCODE                    EXT FIELD = SUPPLIER-ZIPCODE        PHY FIELD = E$SUZIPC FILE = E$SU RC =

USERS
-----
CINCOM
EDUCATION
STUDENT
TJD
```

Programmer's report

The Programmer's Report provides a programmer with all necessary information about a view. Both COBOL and PL/1 programmers can produce this report. The Programmer's Report provides this information for each column in the view:

- ◆ View name
- ◆ Field (Column) type
 - KEY
 - NONUNIQUE KEY
 - CONST
 - REQUIRED
- ◆ Field (Column) name
- ◆ Field type declaration
 - The picture clause (generated by the RDM COBOL preprocessor)
 - The DECLARE clause (generated by the RDM PL/1 preprocessor)
- ◆ Description—Any text information

The programmer uses the column type information when constructing keyed **GET**s or in determining which columns to subset when creating a user view.

The DBA can include any type of text information in defining the view, for example, to give special instructions on how to use the view. The following code listing shows a COBOL Programmer's Report. A PL/1 Programmer's Report follows this example.

SUPRA RDM DIRECTORY REPORTS LEVEL nnnn COPYRIGHT 19nn CSI-ALL RIGHTS RESERVED 14:46:39 04-01-1991 PAGE 329

*** RELATIONAL DATA MANAGER COBOL PROGRAMMER'S REPORT FOR SCHEMA BURRYSCH ***

LOGICAL VIEW: SUPPLIERS-BY-PRODUCT

FIELD TYPE	FIELD NAME	PICTURE	DESCRIPTION
KEY	PRODUCT-CODE	X(009)	BURRYS PRODUCT IDENTIFIER
	PRODUCT-DESC	X(030)	BURRYS PRODUCT IDENTIFIER
	PRODUCT-PRICE	9(07)V9(02)	BURRYS PRODUCT IDENTIFIER
	PRODUCT-WH-QNTY	9(05)	BURRYS PRODUCT IDENTIFIER
	VS-NO-SUPPLIER	X(006)	BURRYS PRODUCT IDENTIFIER
	VS-NO-PART-NO	X(020)	BURRYS PRODUCT IDENTIFIER
	VS-NO-PART-COST	9(07)V9(02)	BURRYS PRODUCT IDENTIFIER
	SUPPLIER-NAME	X(020)	BURRYS PRODUCT IDENTIFIER
	SUPPLIER-ADDR	X(020)	BURRYS PRODUCT IDENTIFIER
	SUPPLIER-CITY	X(013)	BURRYS PRODUCT IDENTIFIER
	SUPPLIER-STATE	X(002)	BURRYS PRODUCT IDENTIFIER
	SUPPLIER-ZIPCODE	9(05)	BURRYS PRODUCT IDENTIFIER

SUPRA RDM DIRECTORY REPORTS LEVEL nnnn COPYRIGHT 19nn CSI-ALL RIGHTS RESERVED 14:46:39 04-01-1991 PAGE 330

*** RELATIONAL DATA MANAGER COBOL PROGRAMMER'S REPORT FOR SCHEMA BURRYSCH ***

LOGICAL VIEW: SUPPLIERS-BY-PRODUCT-VV

FIELD TYPE	FIELD NAME	PICTURE	DESCRIPTION
KEY	PRODUCT-CODE	X(009)	BURRYS PRODUCT IDENTIFIER
	PRODUCT-DESC	X(030)	BURRYS PRODUCT IDENTIFIER
	PRODUCT-PRICE	9(07)V9(02)	BURRYS PRODUCT IDENTIFIER
	PRODUCT-WH-QNTY	9(05)	BURRYS PRODUCT IDENTIFIER
KEY	VS-NO-SUPPLIER	X(006)	BURRYS PRODUCT IDENTIFIER
	VS-NO-PART-NO	X(020)	BURRYS PRODUCT IDENTIFIER
	VS-NO-PART-COST	9(07)V9(02)	BURRYS PRODUCT IDENTIFIER
	SUPPLIER-NAME	X(020)	BURRYS PRODUCT IDENTIFIER
	SUPPLIER-ADDR	X(020)	BURRYS PRODUCT IDENTIFIER
	SUPPLIER-CITY	X(013)	BURRYS PRODUCT IDENTIFIER
	SUPPLIER-STATE	X(002)	BURRYS PRODUCT IDENTIFIER
	SUPPLIER-ZIPCODE	9(05)	BURRYS PRODUCT IDENTIFIER

SUPRA RDM DIRECTORY REPORTS LEVEL nnnn COPYRIGHT 19nn CSI-ALL RIGHTS RESERVED 14:46:39 04-01-1991 PAGE 497

*** RELATIONAL DATA MANAGER PL/1 PROGRAMMER'S REPORT FOR SCHEMA BURRYSCH ***

LOGICAL VIEW: SUPPLIERS-BY-PRODUCT

FIELD TYPE	FIELD NAME	DECLARE	DESCRIPTION
KEY	PRODUCT-CODE	CHAR(9)	BURRYS PRODUCT IDENTIFIER
	PRODUCT-DESC	CHAR(30)	BURRYS PRODUCT IDENTIFIER
	PRODUCT-PRICE	PIC '(7)9V99'	BURRYS PRODUCT IDENTIFIER
	PRODUCT-WH-QNTY	PIC '(4)99'	BURRYS PRODUCT IDENTIFIER
	VS-NO-SUPPLIER	CHAR(6)	BURRYS PRODUCT IDENTIFIER
	VS-NO-PART-NO	CHAR(20)	BURRYS PRODUCT IDENTIFIER
	VS-NO-PART-COST	PIC '(7)9V99'	BURRYS PRODUCT IDENTIFIER
	SUPPLIER-NAME	CHAR(20)	BURRYS PRODUCT IDENTIFIER
	SUPPLIER-ADDR	CHAR(20)	BURRYS PRODUCT IDENTIFIER
	SUPPLIER-CITY	CHAR(13)	BURRYS PRODUCT IDENTIFIER
	SUPPLIER-STATE	CHAR(2)	BURRYS PRODUCT IDENTIFIER
	SUPPLIER-ZIPCODE	PIC '(4)99'	BURRYS PRODUCT IDENTIFIER

SUPRA RDM DIRECTORY REPORTS LEVEL nnnn COPYRIGHT 19nn CSI-ALL RIGHTS RESERVED 14:46:39 04-01-1991 PAGE 498

*** RELATIONAL DATA MANAGER PL/1 PROGRAMMER'S REPORT FOR SCHEMA BURRYSCH ***

LOGICAL VIEW: SUPPLIERS-BY-PRODUCT-VV

FIELD TYPE	FIELD NAME	DECLARE	DESCRIPTION
KEY	PRODUCT-CODE	CHAR(9)	BURRYS PRODUCT IDENTIFIER
	PRODUCT-DESC	CHAR(30)	BURRYS PRODUCT IDENTIFIER
	PRODUCT-PRICE	PIC '(7)9V99'	BURRYS PRODUCT IDENTIFIER
	PRODUCT-WH-QNTY	PIC '(4)99'	BURRYS PRODUCT IDENTIFIER
KEY	VS-NO-SUPPLIER	CHAR(6)	BURRYS PRODUCT IDENTIFIER
	VS-NO-PART-NO	CHAR(20)	BURRYS PRODUCT IDENTIFIER
	VS-NO-PART-COST	PIC '(7)9V99'	BURRYS PRODUCT IDENTIFIER
	SUPPLIER-NAME	CHAR(20)	BURRYS PRODUCT IDENTIFIER
	SUPPLIER-ADDR	CHAR(20)	BURRYS PRODUCT IDENTIFIER
	SUPPLIER-CITY	CHAR(13)	BURRYS PRODUCT IDENTIFIER
	SUPPLIER-STATE	CHAR(2)	BURRYS PRODUCT IDENTIFIER
	SUPPLIER-ZIPCODE	PIC '(4)99'	BURRYS PRODUCT IDENTIFIER

End user report

The End User Report is an abbreviated report intended for the non-technical end user, for example, the SPECTRA user. It provides this information:

- ◆ View name
- ◆ Users related to view
- ◆ Sequence number of each column
- ◆ Column name
- ◆ Column description (from the short text on the Directory)

The following code sample shows an End User Report:

SUPRA RDM DIRECTORY REPORTS LEVEL nnnn COPYRIGHT 19nn CSI-ALL RIGHTS RESERVED 14:46:39 04-01-1991 PAGE 653

*** RELATIONAL DATA MANAGER END USER REPORT FOR SCHEMA BURRYSCH ***

USER: TJD

LOGICAL VIEW: SUPPLIERS-BY-PRODUCT

SEQ#	COLUMN NAME	DESCRIPTION
1	PRODUCT-CODE	BURRYS PRODUCT IDENTIFIER
2	PRODUCT-DESC	BURRY'S PRODUCT DESCRIPTION
3	PRODUCT-PRICE	CURRENT SELLING PRICE OF BURRY'S PRODUCT
4	PRODUCT-WH-QNTY	QUANTITY OF PRODUCTS IN WAREHOUSE
5	VS-NO-SUPPLIER	SUPPLIER(VENDOR) IDENTIFIER
6	VS-NO-PART-NO	VENDOR'S PRODUCT IDENTIFIER
7	VS-NO-PART-COST	VENDOR'S PRODUCT COST
8	SUPPLIER-NAME	NAME OF SUPPLIER
9	SUPPLIER-ADDR	STREET ADDRESS OF SUPPLIER
0	SUPPLIER-CITY	CITY LOCATION OF SUPPLIER
1	SUPPLIER-STATE	STATE WHERE SUPPLIER RESIDES
2	SUPPLIER-ZIPCODE	POSTAL LOCALE OF SUPPLIER

SUPRA RDM DIRECTORY REPORTS LEVEL nnnn COPYRIGHT 19nn CSI-ALL RIGHTS RESERVED 14:46:39 04-01-1991 PAGE 654

*** RELATIONAL DATA MANAGER END USER REPORT FOR SCHEMA BURRYSCH ***

USER: TJD

LOGICAL VIEW: SUPPLIERS-BY-PRODUCT-VV

SEQ#	COLUMN NAME	DESCRIPTION
1	PRODUCT-CODE	DERIVED FROM VIEW PROD
2	PRODUCT-DESC	DERIVED FROM VIEW PROD
3	PRODUCT-PRICE	DERIVED FROM VIEW PROD
4	PRODUCT-WH-QNTY	DERIVED FROM VIEW PROD
5	VS-NO-SUPPLIER	DERIVED FROM VIEW VSNO
6	VS-NO-PART-NO	DERIVED FROM VIEW VSNO
7	VS-NO-PART-COST	DERIVED FROM VIEW VSNO
8	SUPPLIER-NAME	DERIVED FROM VIEW SUPP
9	SUPPLIER-ADDR	DERIVED FROM VIEW SUPP
10	SUPPLIER-CITY	DERIVED FROM VIEW SUPP
11	SUPPLIER-STATE	DERIVED FROM VIEW SUPP
12	SUPPLIER-ZIPCODE	DERIVED FROM VIEW SUPP

Impact of change report

The Impact of Change Report reports any changes you make to files or base views that can impact derived views, application programmers, or SPECTRA users. The Impact of Change Report is divided into three separate reports:

- ◆ Files Impacting Views
- ◆ Views Impacting Views
- ◆ Views Impacting Programs

Files impacting views report

The Files Impacting Views report describes physical changes to files that may impact base or derived views. This report contains the following information:

- ◆ Physical file name
- ◆ Record code
- ◆ Physical Field name
- ◆ External Field name
- ◆ Column name
- ◆ View Name
- ◆ D—File directly impacts this view
- ◆ I—File indirectly impacts this view (the column is derived from another view)

The following code sample shows a Files Impacting Views Report:

TISXA RDM DIRECTORY REPORTS COPYRIGHT 19nn CSI - ALL RIGHTS RESERVED 16:13:35 06/02/87

*** LOGICAL VIEW IMPACT OF CHANGE REPORT FOR FILES IMPACTING VIEWS

FILE : E\$PD

RECORD CODE	PHYSICAL FIELD	EXTERNAL FIELD	COLUMN NAME	VIEW NAME	TYPE OF IMPACT
E\$PDCTRL	PRODUCT-CODE		INVLIN-PRODUCT	ADD-INVOICE-VV	I
			INVLIN-PRODUCT	INVL	D
			INVLIN-PRODUCT	INVOICE-VV	I
			INVLIN-PRODUCT	VV-INV-VV	I
			MANLINE-PRODUCT	MANIFEST-VV	I
			MANLINE-PRODUCT	MANL	D
			PRODUCT-CODE	BILL	D
			PRODUCT-CODE	BRANCH-STOCK-BY-PRODUCT	D
			PRODUCT-CODE	BRANCH-STOCK-BY-PRODUCT-VV	I
			PRODUCT-CODE	MAIN-WAREHOUSE-INVEN-VV	I
			PRODUCT-CODE	MAIN-WAREHOUSE-INVENTORY	D
			PRODUCT-CODE	PROD	D
			PRODUCT-CODE	PROD-SUPP-VV	I
			PRODUCT-CODE	PRODUCT-PURCHASE-INFO-VV	I
			PRODUCT-CODE	SUPPLIERS-BY-PRODUCT	D
			PRODUCT-CODE	SUPPLIERS-BY-PRODUCT-VV	I
			PRODUCT-CODE	VERIFY-PRODUCT	D
			PRODUCT-CODE	VERIFY-PRODUCT-VV	I
			PRODUCT-CODE	VV-PROD	I
			STOCK-PRODUCT	BRANCH-STOCK-VV	I
			STOCK-PRODUCT	REGIONAL-SHIPPING-VV	I
			STOCK-PRODUCT	STCK	D
			STOCK-PRODUCT	UPDATE-STOCK-VV	I
			STRUCTURE-ASSM	MAIN-WAREHOUSE-INVEN-VV	I
			STRUCTURE-ASSM	STRU	D
			VS-NO-PRODUCT	VENDOR-STOCK-NUMS-VV	I
			VS-NO-PRODUCT	VSNO	D
E\$PDDESC	PRODUCT-DESC		PRODUCT-DESC	ADD-INVOICE-VV	D
			PRODUCT-DESC	BRANCH-STOCK	D
			PRODUCT-DESC	BRANCH-STOCK-BY-PRODUCT	D
			PRODUCT-DESC	BRANCH-STOCK-BY-PRODUCT-VV	I
			PRODUCT-DESC	BRANCH-STOCK-VV	I
			PRODUCT-DESC	INVOICE	D
			PRODUCT-DESC	INVOICE-VV	I
			PRODUCT-DESC	MAIN-WAREHOUSE-INVEN-VV	I
			PRODUCT-DESC	MAIN-WAREHOUSE-INVENTORY	D
			PRODUCT-DESC	MANIFEST	D
			PRODUCT-DESC	MANIFEST-VV	I
			PRODUCT-DESC	PO-BY-DATE	D
			PRODUCT-DESC	PROD	D

Views impacting views report

The Views Impacting Views Report describes changes to base views that impact derived views. This report contains the following information:

- ◆ Impacting view name (base view)
- ◆ Impacted view name
- ◆ D—Impact is direct
- ◆ I—Impact is indirect

If a view is not derived from any other views, the following information appears on the report:

- ◆ View name
- ◆ Message: NO IMPACTING VIEWS

Views derived from views that do not themselves impact views are not listed.

The following listing shows a Views Impacting Views Report. In the example, impact is direct (D) for all views beginning with ADD-INVOICE-VV through VERIFY-PRODUCT-VV. The type of impact prints on the report only when it changes. If the TYPE OF IMPACT field is blank for a particular view, assume it is the same as the most recent impact printed.

TIS/XA RDM DIRECTORY REPORTS COPYRIGHT 19nn CSI - ALL RIGHTS RESERVED 16:13:35 06/02/87

*** LOGICAL VIEW IMPACT OF CHANGE REPORT FOR VIEWS IMPACTING VIEWS

VIEW : PROD

IMPACTED VIEW NAME	TYPE OF IMPACT
NO IMPACTING VIEWS	
ADD-INVOICE-VV	D
BRANCH-STOCK-BY-PRODUCT-VV	
BRANCH-STOCK-VV	
INVOICE-VV	
MAIN-WAREHOUSE-INVEN-VV	
MANIFEST-VV	
PROD-SUPP-VV	
PRODUCT-PURCHASE-INFO-VV	
REGIONAL-SHIPPING-VV	
SUPPLIERS-BY-PRODUCT-VV	
UPDATE-STOCK-VV	
VENDOR-STOCK-NUMS-VV	
VERIFY-PRODUCT-VV	
VV-INV-VV	I
VV-PROD	D

Views impacting programs report

The Views Impacting Programs report is indexed by view and lists all programs that use a view. When you change a view definition, use this report to find any affected programs. This report contains the following information:

- ◆ Impacting view name
- ◆ Program name
- ◆ D—Impact is direct
- ◆ I—Impact is indirect

Views used by programs report

When an RDM processor processes applications, they are enrolled in the Directory and automatically related to the views they use. The Views Used by Programs Report provides a list of programs each view uses and the following information:

- ◆ View name
- ◆ All programs that use that name
- ◆ Date the view was last updated
- ◆ Time the view was last updated

The following listing shows a Views Used by Programs Report:

```
SUPRA RDM DIRECTORY REPORTS  LEVEL nnnn  COPYRIGHT 19nn CSI - ALL RIGHTS RESERVED 14:46:39
04-01-1991
PAGE 695
*** RELATIONAL DATA MANAGER VIEWS USED BY PROGRAMS REPORT FOR SCHEMA BURRYSCH ***

LOGICAL VIEWS          PROGRAMS          DATE UPDATED      TIME UPDATED
SUPPLIERS-BY-PRODUCT  **** NO PROCEDURES RELATED TO THIS LOGICAL VIEW ****
SUPPLIERS-BY-PRODUCT-VV **** NO PROCEDURES RELATED TO THIS LOGICAL VIEW ****
```


9

Configuring the RDM for your environment

Overview of configuring the RDM for your environment

This chapter tells you how to find the information to configure and use RDM in your operating environment (under the operating system and teleprocessing monitor you use). This chapter also lists the RDM modules by operating system and summarizes the new and different RDM characteristics.

Your SUPRA Server libraries contain procedures and job control language (JCL) samples for running RDM-related jobs in your operating environment. Samples are subject to change. See the SUPRA Server JCL library or source statement library member TXJ\$INDX for a list of JCL samples.

OS/390

See the SUPRA Server procedure library member TIS\$RDM for a list of RDM procedures. See the SUPRA Server macro library or member TX\$\$INDX for an index to the different kinds of samples.

VSE

See the SUPRA Server RDM sublibrary member TXJ\$INDX for a list of JCL. The sample job to assemble and link the options module can be found in that list.

See your *SUPRA Server Installation Guide* for the latest information relevant to your installation, and for specific instructions on linking, initializing, and bringing up RDM the first time.

Refer to the *SUPRA Server OS/390 Installation Guide*, P26-0149, or the *SUPRA Server VSE Installation Guide*, P26-0132, for information on the resources (memory and disk space) requirements and usage for RDM and other SUPRA Server components under your operating environment.

See “[Setting the online RDM options with macros](#)” on page 261 for information on how to set RDM memory options and other options with the RDM macro C\$VOOPTM.

You must process your COBOL or PL/1 RDM application program source code with the RDML precompiler before you compile the code with the standard COBOL or PL/1 compiler. Refer to the RDM programming guides for information on creating, precompiling, compiling, linking, and running RDM application programs:

- ◆ *SUPRA Server PDM RDM COBOL Programming Guide (OS/390 & VSE)*, P26-8330
- ◆ *SUPRA Server PDM RDM PL/1 Programming Guide (OS/390 & VSE)*, P26-8331

The implementation of online RDM under OS/390/XA, OS/390/ESA or VSE/ESA differs from other versions of RDM in that it takes advantage of extended memory (memory above the 16 MB line):

- ◆ It allocates task work memory in separate heaps and stacks rather than in slots. It allows the heaps to be allocated in extended memory. (One heap and one stack are the equivalent of one slot. One active task employs one heap and one stack. The stack is in storage only for the life of an RDML command.)
- ◆ It allows the allocation of global views in extended memory.
- ◆ It loads the largest part of RDM, the resident module (CSVNVRES), in extended memory. The table on the following page lists and briefly describes each of the major modules in RDM.

OS/390 online load module	OS/390 batch load module	VSE online phase	VSE batch phase	Description
CSVCOBPP	CSVCOBPP	CSVCOBPP	CSVCOBPP	COBOL preprocessor
CSVPL1PP	CSVPL1PP	CSVPL1PP	CSVPL1PP	PL/1 preprocessor
n/a	CSVIBDBA	n/a	CSVJBDBA	Batch DBAID Support
n/a	CSVIBINT	n/a	CSVJBINT	Batch RDM Support
CSVLVDBA	CSVLVDBA	CSVLVDBA	CSVLVDBA	DBAID mainline
CSVNVRES	CSVLVRES	CSVNVRES	CSVLVRES	Reentrant component of RDM mainline
CSVNVRUN	CSVLVRUN	CSVNVRUN	CSVLVRUN	Nonreentrant component of RDM mainline
CSVODBA	n/a	CSVODBA	n/a	CICS DBAID Support
CSVOPLVS	n/a	CSVOPLVS	n/a	CICS Program Interface
CSVNDATB	n/a	CSVNDATB	n/a	CICS PDM Support
CSVNVSAM	n/a	CSVNVSAM	n/a	VSAM Support—CICS
n/a	CSVIVSAM	n/a	CSVJVSAM	VSAM Support—Batch
n/a	CSVUREPT	n/a	CSVUREPT	RDM Directory Report Mainline
n/a	CSVILUV CSVOSVS (Alias) CSVIOSVS (Alias)	n/a	CSVJLUV	Batch Program Interface
CSVNICIC	n/a	CSVNICIC	n/a	CICS Program Interface
CSVNPLVS	n/a	CSVNPLVS	n/a	CICS RDM Support
CSVNRDIN	n/a	CSVNRDIN	n/a	RDM START/STOP Processing
CSVXRSSO	n/a	CSVXRSSO	n/a	RESET/SINOF Processing

Configuring the RDM XA storage

Task context is allocated in a separate heap and stack. You specify heaps and stacks in the options module, CSVOOPTM (OS/390) or CSVDOPTM.A (VSE). See [“Customizing the RDM processing with user exits”](#) on page 237 for information about coding these options.

Heaps store view context and can be allocated above the 16 MB line. Heap storage is required for every RDM CICS transaction, and is retained until the transaction signs off of RDM. All heap storage is allocated during RDM initialization. The size and number of heaps is determined by the options module CSVOOPTM (OS/390) or CSVDOPTM.A (VSE). If more tasks sign on to RDM than there are available heaps, heap storage is rolled to auxiliary temporary storage for pseudoconversational tasks.

Stacks store task context for the PASCAL routines of RDM. Stacks are always allocated below the 16 MB line. Stack storage is required for every RDM CICS task. The storage is allocated when the task issues its first RDML and is released when the task either issues an RDML sign-off or detaches. Stacks are allocated from the CICS dynamic storage area (DSA).

The global view pool size is also defined in the options module CSVOOPTM or CSVDOPTM.A. Since the pool can be allocated above the 16 MB line, you should specify a size large enough to accommodate the number of views you want to remain in memory. After global views are opened, all unused memory in the pool is released.

Heaps and the global view pool are acquired from storage managed by the operating system (GETMAIN with OS/390, GETVIS with VSE). The available space will be the size of your CICS region minus the DSASZE operand you specify in the System Initialization Table (SIT). Refer to the [SUPRA Server PDM CICS Connector Systems Programming Guide \(OS/390 & VSE\)](#), P26-7452, for additional information about calculating storage.

The options module also defines whether to allocate RDM CICS storage above or below the 16 MB line. If you run the Physical Data Manager (PDM) in the attached mode, you must allocate RDM CICS storage below the line. Refer to the [SUPRA Server PDM Tuning Guide \(OS/390 & VSE\)](#), P26-0225, for additional information about defining task and global view storage.

Interaction of options parameters

The various SUPRA Server options in a CICS environment include the interaction of parameters among the following:

- ◆ The SUPRA Server active environment description
- ◆ The SUPRA Server CICS Connector Options table
- ◆ The SUPRA Server CICS Connector OPER CONNECT request
- ◆ The RDM C\$VOOPTM macro and the options module
- ◆ The IBM CICS system SIT values

Environment description parameters

A central PDM operates under control of an active environment description. You identify this active environment description in the REALM parameter of the CSIPARM file (ENVDESC=). The active environment description supplies three values:

- ◆ **Maximum Connected Interfaces value.** This value sets the limit for the maximum number of regions/partitions that can communicate with a central PDM. PDM connects one interface for each region/partition when it identifies itself via a sign-on or connect request. An interface remains connected until the region/partition severs the connection. A CICS system uses one interface.
- ◆ **Maximum Signed On Tasks value.** This value sets the total number of tasks that the central PDM can service throughout the system.
- ◆ **Maximum Connected Threads value.** This value sets the limit on the number of concurrent PDML requests that PDM will service. The PDM allocates a thread to a PDM task while a PDML is being serviced. The thread is freed when the PDML completes.

For each CICS interface, a THREADS parameter can be specified on the OPER CONNECT request. This number represents a subset of the Maximum Connected Threads in the active environment description.

In a CICS system, the default value for THREADS is the CSTXOPRM THREADS parameter. (In the following discussion when the CSTXOPRM THREADS value is mentioned, it is understood that a given OPER CONNECT can set a different value.) This parameter value sets the maximum number of PDMLs the PDM can service concurrently for that CICS system. When CICS issues its OPER CONNECT, PDM attempts to reserve this number of threads for that interface. If the limit of PDM threads from the active environment description has been exceeded, the OPER CONNECT fails with a PDM IPAR status. The threads in a CICS interface remain reserved for that CICS until CICS issues an OPER DISCONNECT which releases the threads as well as the tasks and interface.

The connect/sinon process

In a CICS environment, the CICS system must connect its interface to the PDM before any CICS transaction is allowed to issue any PDML calls. Before the interface is connected, every PDML issued by a CICS transaction will receive a PDM NOTO status instead of the requested service. The connection of the interface occurs in the CICS Connector component of SUPRA Server; the connect request is invoked either automatically, using standard CICS initialization features, or manually, after CICS has initialized. Both methods involve issuing the OPER transaction with the CONNECT subfunction. OPER CONNECT connects one interface between a CICS and a central PDM. If the number of connected interfaces in the system reaches the Maximum Connected Interfaces value, the OPER CONNECT request receives a PDM IPAR status instead of connecting the interface.

OPER CONNECT parameters

An OPER CONNECT request specifies a TASKS parameter. The default value for this parameter is the CSTXOPRM TASKS parameter. (CSTXOPRM is the macro for generating the SUPRA Server CICS Connector CSTXOTBL module. However, a given OPER CONNECT can set a different value than is defined in CSTXOPRM.) When PDM services an OPER CONNECT, it attempts to reserve a number of task entries as specified in the TASKS parameter for the CICS interface. The TASKS parameter value sets the maximum number of tasks allowed to sign on to the PDM in that CICS region. This is the maximum number of transactions in a CICS that can issue a PDML SINON without issuing a matching PDML SINOF. If the number of PDM tasks signed on or reserved exceeds the maximum signed on tasks value in the active environment description, the OPER CONNECT fails with a PDM IPAR status. Once an OPER CONNECT succeeds, transactions within a CICS can sign on to the PDM, up to the limit imposed by CSTXOPRM TASKS. Once this limit is reached, the next transaction within that CICS attempting a PDML SINON receives a PDM CFUL status. A CICS interface remains connected and all its PDM tasks remain reserved until the interface is disconnected. The OPER DISCONNECT releases all the tasks reserved for that interface and disconnects the interface.

RDML processing

If a CICS transaction issues RDMLs, the first RDML must be an RDML sign-on (either implicit or explicit), which in turn issues a PDML SINON. This means that every RDM transaction is one of the PDM tasks. The limit on RDM CICS tasks is the C\$VOOPTM RDMUSR# parameter. The maximum value of RDMUSR# is CSTXOPRM minus any task requirements for non-RDM tasks. However, if RDMUSR# is greater than TASKS, the number of signed-on RDM transactions can never reach its maximum of RDMUSR#. Thus, if RDMUSR# is less than TASKS and the number of signed-on RDM transactions reaches RDMUSR#, the next transaction to issue an RDML sign-on will receive a failure status due to insufficient resources to service the request. And if RDMUSR# is greater than TASKS and the number of signed-on RDM transactions reaches TASKS, the next transaction to issue an RDML sign-on will receive a failure status due to a PDM CFUL status.

Each RDM task triggers the allocation of stack storage from CICS DSA below the 16 MB line. A stack exists until the task signs off RDM or detaches, at which time the stack storage is released to CICS DSA. Since the limit on concurrent RDM transactions is controlled by the CICS parameter AMXT (CICS 3.3 and below) or MXT (CICS 4 and above), this number is also the limit on the number of stacks that can be allocated at the same time. Stack size is controlled by the C\$VOOPTM STACKSZ parameter. A value of STACKSZ=32K should be sufficient for most environments.

During RDM initialization, several RDM storage areas (heaps) are allocated from virtual storage below or above the 16 MB line. The C\$VOOPTM HEAP# parameter determines the number of heaps, the C\$VOOPTM HEAPSZ parameter determines the size of each heap, and the C\$VOOPTM GETMAIN parameter determines the location above or below the line. The heaps remain allocated throughout the CICS execution.



The value GETMAIN=A is available only if you purchase an extra-cost option. Please contact your Cincom representative for more information.

When a transaction issues an RDML sign-on, RDM assigns a heap to that transaction. The heap remains assigned to that transaction until it issues an RDML sign-off or terminates abnormally, at which time RDM releases the heap, making it available to other RDM transactions. When a signed-on RDM transaction opens a logical view, RDM builds the context areas for the open view in that transaction's heap. When CICS passes control to an RDM transaction, it must have both a stack and a heap available to it. If the RDM transaction is conversational, the end of the transaction triggers an RDML sign-off (if the application code has not already issued one), and the stack and heap are released. But if the RDM transaction is pseudoconversational, a task within the application can end without issuing an RDML sign-off. In that case, RDM manages the stack and heap differently. As explained above, the stack is simply freed, but pseudoconversational transactions must be able to access their views in each task making up the application. So a view opened in one task of a pseudoconversation may still be open in later tasks of the application. For this reason, the contents of an RDM application's heap must be available to all tasks making up the application. RDM preserves the contents of the heap for the next task in the application. There must therefore be as many heaps as signed-on RDM applications, which can be as high as the C\$VOOPTM RDMUSR# value.

It is not always practical to allocate this many heaps. You can conserve storage by setting HEAP# less than RDMUSR# but there is a performance trade-off in doing so. When you set these two parameters this way, RDM writes heaps out to CICS temporary storage and reads them back in, so that active, executing RDM transactions can share the heaps. (This I/O is referred to as heap roll-out and roll-in.) When a task within an RDM pseudoconversational application ends, RDM can reassign its heap to another RDM transaction that signs on or resumes. First the heap is rolled out, then it is reassigned to the new transaction. If this transaction is resuming, its heap is then rolled in.

PDM thread processing

A PDM thread is allocated to a CICS task in the process of having a PDML serviced, so you should set THREADS no higher than TASKS in a given CICS interface. If your CICS has a very high transaction rate, you might observe frequent TFUL statuses. To resolve this problem, you should increase the CSTXOPRM THREADS value. While a PDML is being serviced on behalf of a PDM task, its active CICS task is in use. The limit on the number of concurrent CICS active tasks is the DFHSIT AMXT (or MXT) value. Therefore, CSTXOPRM THREADS should not be less than AMXT (or MXT). If you set THREADS less than AMXT (or MXT) and the number of PDMLs being concurrently serviced in that CICS reaches the THREADS value, the next PDML attempted fails with a TFUL status. Similarly, since each active RDM task requires a thread, you should not set HEAP# less than the value of THREADS. If you do, then once the number of active RDM tasks reaches HEAP#, the next task attempting an RDML or resuming fails, due to insufficient resources for service.

CICS processing

When any CICS transaction requests a CICS service, the CICS dispatcher can suspend the transaction if the overall CICS load is heavy. Suspension is unlikely to occur in a pseudoconversational application, since its tasks terminate often, which frees their threads. But a conversational transaction retains its active CICS task throughout the life of the transaction, so long as it is active. Therefore, if all active CICS tasks are allocated, suspension of a conversational transaction is more likely.

A suspended transaction remains signed on to CICS, although it loses its active CICS task while it is suspended. If the suspended transaction is a PDM task, the PDM maintains this transaction as a signed-on task. If the suspension occurs at a PDML, PDM keeps the suspended transaction's PDM thread active as well, even though its active CICS task has been lost. The PDM thread remains active for the duration of the suspension and afterwards until the PDML completes, at which time the PDM thread is freed. If another PDM task in that CICS issues a PDML during the suspension, it is possible that CICS will require more active PDM threads than the DFHSIT AMXT (or MXT) value.

You should therefore set CSTXOPRM THREADS slightly higher than DFHSIT AMXT (or MXT), to allow for task suspension. Typically, $AMXT+1 = THREADS$ or $AMXT+2 = THREADS$ works well. But if your CICS has a very high transaction rate, you might observe frequent TFUL statuses. To resolve this problem, increase the CSTXOPRM THREADS value.

CICS limits

The limit CICS imposes on the number of concurrently active and executing transactions is the DFHSIT AMXT (or MXT) value. If all of these are RDM transactions, this means that C\$VOOPTM RDMUSR# should not be less than DFHSIT AMXT (or MXT). If you set RDMUSR# less than AMXT (or MXT) and the number of active, executing RDM transactions reaches RDMUSR#, the next new or resuming RDM transaction that attempts an RDML will receive a failure status due to insufficient resources for servicing the RDML.

A

Customizing the RDM processing with user exits

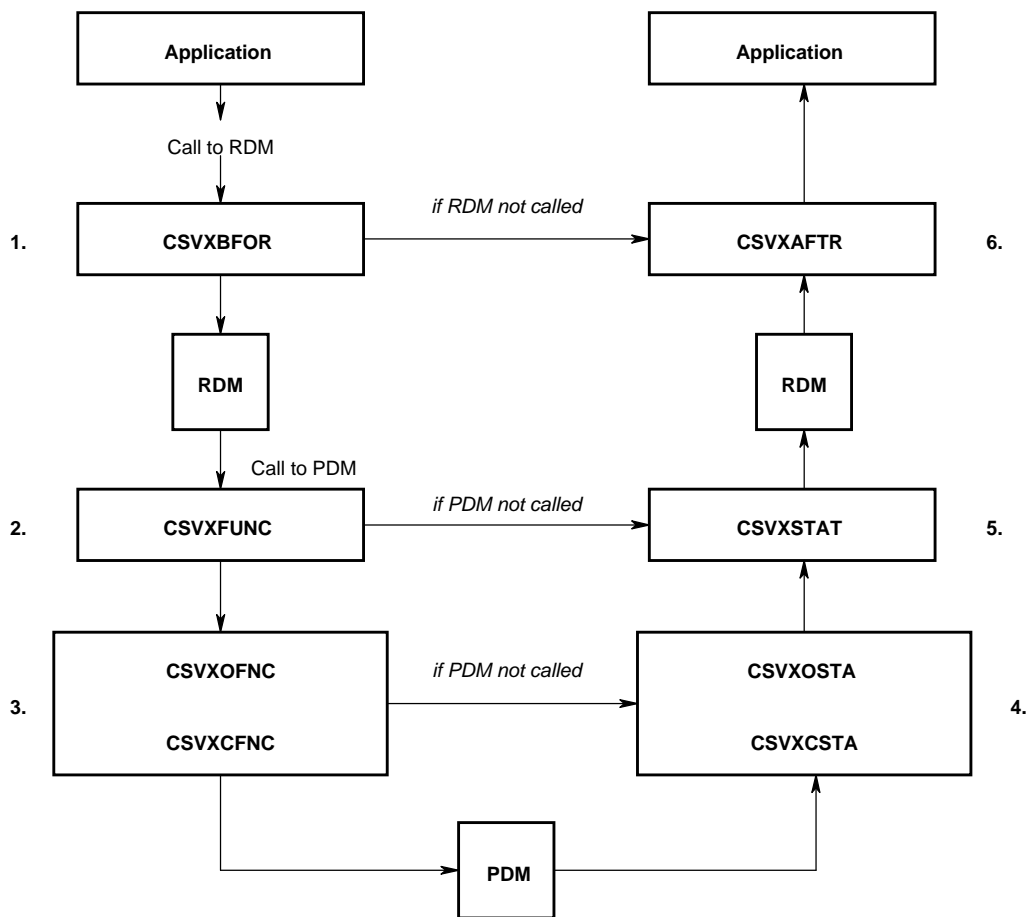
Overview of customizing the RDM processing with user exits

Several exits are available to RDM users. These exits allow you to insert processing routines before and after database or RDML calls and to perform validation checking. You can use the database exits and RDML exits to bypass database or RDML calls, to perform your own database or user file calls, or to satisfy any special requirements for your system. You can use the validation exits to perform complex validation logic.

This appendix describes the following types of RDM user exits:

- ◆ “Using database exits” on page 240 describes the database exits: environment-independent and environment-dependent variations of the function exit and the status exit.
- ◆ “Using RDML exits” on page 249 describes the RDML exits: the before-function exit and the after-function exit.
- ◆ “Using validation exits” on page 256 describes validation exits.

The following figure illustrates the location and control flow of the RDM exit points. In the following explanation, a number in parentheses refers to the corresponding number in the diagram. If you do not provide an exit interface at an exit point, processing continues as though that exit point does not exist.



Order of events during call to RDM:

1. Application issues RDML command.
2. Control passes to CSVXBFOR (1) and invokes exit interface.
3. If CSVXBFOR denies access to the RDM, then control passes to CSVXAFTR (see Step 13), else control passes to the RDM.
4. The RDM issues a call to the PDM.
5. Control passes to CSVXFUNC (2) and invokes exit interface.
6. If CSVXFUNC denies access to the PDM, then control passes to CSVXSTAT (see Step 11), or else control passes directly to one of the exit points listed in Step 7.
7. Control passes to one of the following environment-dependent exit points:

```
For CICS -- CSVXCFNC (3)
For batch -- CSVXOFNC (3)
```
8. If CSVXCFNC or CSVXOFNC denies access to the PDM, then control passes to one of the environment-dependent exit points listed in Step 10, or else control passes to the PDM.
9. The PDM passes control back to the RDM.
10. Control passes to one of the following environment-dependent exit points:

```
For CICS -- CSVXCSTA (4)
For batch -- CSVXOSTA (4)
```
11. Control passes to CSVXSTAT (5) and invokes exit interface.
12. Control passes to the RDM.
13. Control passes to CSVXAFTR (6) and invokes exit interface.
14. Control passes back to the application.

The following table shows RDM user exit programs and the addressing mode in which receive control in a CICS environment:

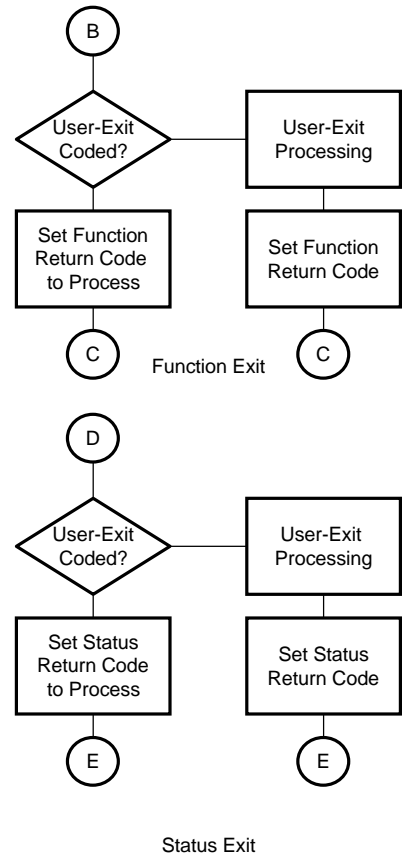
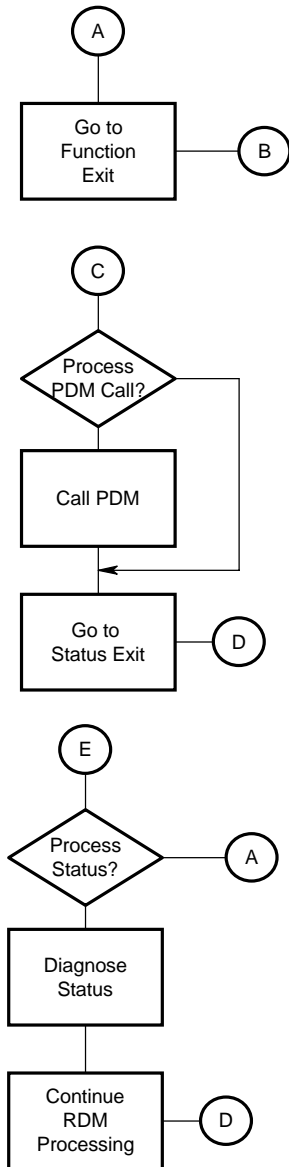
Module	Addressing mode
CSVXAFTTR	24-bit mode
CSVXBFOR	24-bit mode
CSVXCFNC	Same addressing mode as the invoking user application
CSVXCSTA	Same addressing mode as the invoking user application
CSVXFUNC	24-bit mode
CSVXSTAT	24-bit mode
CSVXCVXT	24-bit mode or 31-bit mode*

* If CSVXCVXT is link edited with CSVNPLVS, the exit receives control in 31-bit mode. If CSVXCVXT is not linked with CSVNPLVS, it receives control in 24-bit mode (via a CICS command level LINK).

Using database exits

Use the database exits to insert routines at the RDM processing level, which is environment-independent, and/or at the physical data manager (PDM) call processing level, which is environment-dependent. The following figure illustrates the processing flow of the four database exits. The available database exits and their associated names are:

Environment- independent batch			
Exits	Environment-independent	Batch	CICS
Function	CSVXFUNC	CSVXOFNC	CSVXCFNC
Status	CSVXSTAT	CSVXOSTA	CSVXCSTA



Using environment-independent database exits

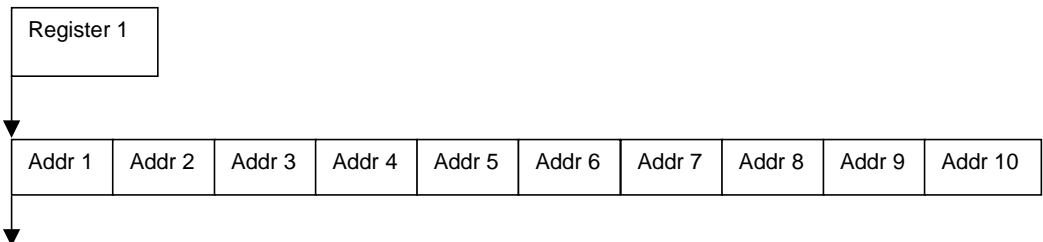
You must always link edit the environment-independent exits (CSVXFUNC and CSVXSTAT) with the RDM run-time processor prior to program execution. The system provides default exits that give the parameter list back to RDM unchanged, meaning to go ahead and process normally. You can write these exits in Assembler, FORTRAN, or COBOL. The contents of the general purpose registers on entry to the exit program are as follows:

Register	Contents
1	Points to the address list of the function request parameters.
13	Contains the address of the standard register save area. You must save and restore all registers that your routine might alter.
14	Contains the standard branch return register address for return to the RDM run-time processor from your exit routine.
15	Contains the address of the user exit. You may use this register to initialize the user exit base register.

Using the function exit (CSVXFUNC)

Use the CSVXFUNC exit to analyze and modify, or bypass, the operation of the physical data manager (PDM). RDM calls CSVXFUNC just before calling the PDM with a physical data manipulation language (DML) function request. Upon entry to CSVXFUNC, register 1 points to an address list of the physical DML function's parameters (see the following figure). The system provides a null exit which gives the parameter list back to RDM unchanged, meaning to go ahead and process. You can code this exit to do your own unsupported file structure processing. A return code of 1 causes RDM to skip the PDM call. RDM proceeds to call the PDM if the return code has any value other than 1. Prior to entry, the RDM run-time processor initializes STATUS to **** and SKIPDBMS to 0.

If you do not bypass the PDM call, you may not alter the FUNC parameter. You may, however, modify any of the other parameters. It is possible to do your own file access, change the data areas, set a return code of 1 in SKIPDBMS, return to the run-time processor, and RDM would bypass the PDM call. If you do, you must set a PDM status code if an error occurs, or leave it as **** if no error occurs, since the run-time processor checks the status to see whether the PDM was called or not. You may use your own 4-character status code. This would be recorded on the console log, and the standard PDM error message would be passed to the application message area. Or, you may use PDM status codes, which RDM would intercept.

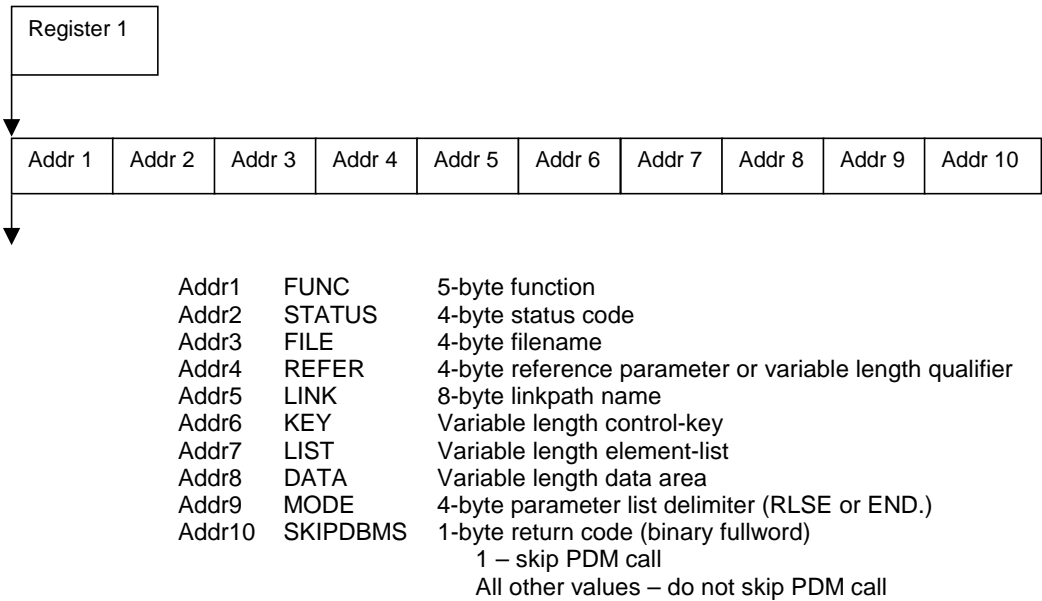


Addr1	FUNC	5-byte function
Addr2	STATUS	4-byte status code
Addr3	FILE	4-byte filename
Addr4	REFER	4-byte reference parameter or variable length qualifier
Addr5	LINK	8-byte linkpath name
Addr6	KEY	Variable length control-key
Addr7	LIST	Variable length element-list
Addr8	DATA	Variable length data area
Addr9	MODE	4-byte parameter list delimiter
Addr10	SKIPDBMS	4-byte return code (binary fullword)
		1 – skip PDM call
		All other values – do not skip PDM call

Using the status exit (CSVXSTAT)

Use the CSVXSTAT exit to analyze the return status from a physical data manager (PDM) function call and to take appropriate action when the PDM returns an unsuccessful status. Upon entry to the exit program, register 1 points to an address list of the physical data manipulation language (DML) function request parameters (see the following figure). The system provides a null exit which gives the parameter list back to RDM unchanged, meaning to go ahead and process normally. You can code this exit to cause RDM to bypass the status code check and try the PDM call again. A return code of 1 indicates that you want RDM to call the PDM again with the same function. Any other value indicates RDM should not repeat the PDM call but continue processing. RDM does not alter any of the parameter values before calling the exit.

Within this exit, you can interrogate the status that was returned either from the function exit (CSVXFUNC) or from the normal PDM call. You can also examine the data and send a return code to RDM depending on the results. A repeat of the PDM call also invokes the CSVXFUNC exit. Therefore, if you have coded the CSVXFUNC exit, it executes again if you set LOOP to 1.



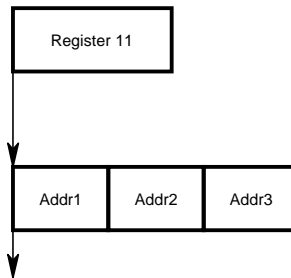
Using environment-dependent database exits

The RDM environment-dependent run-time interface module uses weak external references to the environment-dependent exits. These exits are optional. Use them cautiously and only when necessary. The system provides null exits, but only in source code for use as examples.

You can write these exits only in assembler language because the register conventions are nonstandard. Contents of the general purpose registers on entry to the exit program are as follows:

Register	Batch CSVXOFNC/CSVXOSTA	CICS CSVXCFNC/CSVXCSTA
11	Parameter list	Parameter list
12	-	EIB
13	-	EISTG
14	Return address	Return address
15	Exit entry point	Exit entry point

RDM supplies no register save area to the exits. The RDM restores its registers when the exit returns control to it. The parameter list (pointed to by register 11) consists of a list of three addresses:



The three addresses are as follows:

1. The address of the physical data manager request parameters address list. This list of addresses is variable in length according to the physical data manager request to be processed.
2. The address of the exit return code.
3. The address of a 256-byte work area available to the exit. In a multitasking environment (CICS), the contents of the work area remain unchanged from one physical data manager call to the next until the application issues an RDM **SIGN-ON** or **SIGN-OFF** command. However, the address of the work area may change from call to call; therefore, do not store any addresses that point to the work area. A SIGN-ON or SIGN-OFF command initializes the work area to binary zeros. Note this work area is also passed to the validation exit.

You must link your environment-dependent exits with the appropriate RDM interface modules or phases as follows:

- ◆ Under OS/390:
 - To link CSVXOFNC and/or CSVXOSTA with batch RDM:
Insert an INCLUDE statement for each exit in the link deck CSVIBINT. Put the INCLUDE(s) for your exit(s) anywhere after the INCLUDE for module CSVIVSCI and before the NAME statement. Then use this link deck to relink the module CSVIBINT.
 - To link CSVXCFNC and/or CSVXCSTA with online RDM:
Insert an INCLUDE statement for each exit in the link deck CSVNPLVS. Put the INCLUDE(s) for your exit(s) immediately before the ENTRY statement. Then use this link deck to relink the module CSVNPLVS.
 - To link CSVXOFNC and/or CSVXOSTA with batch DBAID:
Insert an INCLUDE statement for each exit in the link deck CSVIBDBA. Put the INCLUDE(s) for your exit(s) anywhere before the ENTRY statement. Then use this link deck to relink the module CSVIBDBA.
- ◆ Under VSE:
 - To link CSVXOFNC and/or CSVXOSTA with batch RDM:
Insert an INCLUDE statement for each exit in the link deck CSVJBINT on the relocateable library. Put the INCLUDE(s) for your exit(s) anywhere after the PHASE statement and before the ENTRY statement. Then use this link deck to relink the phase CSVJBINT.
 - To link CSVXCFNC and/or CSVXCSTA with online RDM:
Insert an INCLUDE statement for each exit in the link deck for CSVNPLVS. Put the INCLUDE(s) for your exit(s) immediately before the ENTRY statement. Then use this link deck to relink the phase CSVNPLVS.
 - To link CSVXOFNC and/or CSVXOSTA with batch DBAID:
Insert an INCLUDE statement for each exit in the link deck CSVJBDBA on the relocateable library. Put the INCLUDE(s) for your exit(s) anywhere after the PHASE statement and before the ENTRY statement. Then use this link deck to relink the phase CSVJBDBA.

Your SUPRA Server job control language (JCL) library or source statement library contains a sample JCL member named TXJLINK for link editing Cincom software. Samples are subject to change. See the SUPRA Server JCL library or source statement library member TXJ\$INDX for a list showing this and other JCL samples.

OS/390

See the SUPRA Server macro library or source statement library member TX\$IINDX for a list of link decks. See the SUPRA Server macro library or source statement library member TX\$\$INDX for a list of different kinds of samples.

Using the function exit (CSVXOFNC or CSVXCFNC)

RDM invokes the environment-dependent function exit after the environment-independent function exit (CSVXFUNC; see “[Using the function exit \(CSVXFUNC\)](#)” on page 243) and before the physical data manager (PDM) call processing. Use the function exit to analyze and modify (or bypass) the operation of the PDM call. You can code this exit to do your own unsupported file structure processing. RDM skips the PDM call if the exit returns a value of 1 in SKIPDBMS; RDM proceeds to call the PDM if the exit returns any value other than 1. RDM initializes SKIPDBMS to 0 before calling the exit.

Using the status exit (CSVXOSTA or CSVXCSTA)

RDM invokes the environment-dependent status exit after physical data manager (PDM) processing and before the environment-independent status exit (CSVXSTAT; see “[Using the status exit \(CSVXSTAT\)](#)” on page 244). Use the status exit to analyze the status returned from a PDM call and to take the appropriate action. You can also use this exit to return to the PDM and retry an unsuccessful request. A return code (LOOP) of 1 indicates that you should repeat the PDM function. Any other value means that you should not repeat the PDM call but continue processing.

Using RDML exits

The CSVXBFOR and CSVXAFTTR exits were created so you can analyze, modify, or bypass the operation of RDM. RDM invokes CSVXBFOR before performing any RDM function, and invokes CSVXAFTTR after performing any RDM function. These exits allow you to bypass normal sign-on and sign-off procedures and to perform any other functions.

All RDML exits are entered in 24-bit addressing mode (AMODE=24). Your exit code must switch to 31-bit addressing mode if your exit issues commands that require 31-bit addressing capability. If your exit switches to 31-bit addressing mode, it must switch back to 24-bit mode before returning to RDM.

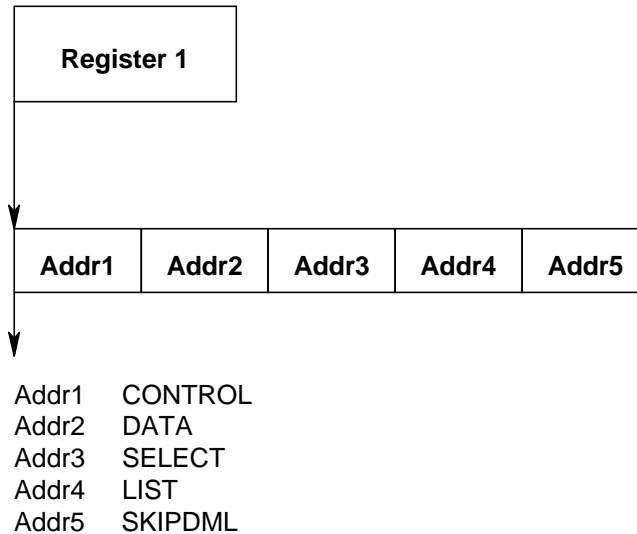
Under VSE/ESA, the AMODE considerations are the same as for OS/390/XA or OS/390/ESA. Under VSE/SP, all exits are entered in AMODE-24. AMODE-31 is not available to VSE/SP.

The system provides default exits which give the parameter list back to RDM unchanged, indicating RDM should process normally. You may write these exits in Assembler, FORTRAN, or COBOL. Contents of the general purpose registers on entry to the exit program are as follows:

Register	Contents
1	Points to the address list of the RDM function request parameters.
13	Contains the address of the standard register save area. You must save and restore all registers that your routine might alter.
14	Contains the standard branch return register address for return to the RDM run-time processor from your exit routine.
15	Contains the address of the user exit. Use this register to initialize the user exit base register.

Using the before-function exit (CSVXBFOR)

Use the CSVXBFOR exit to analyze and modify, or bypass, the operation of RDM. RDM invokes CSVXBFOR just before performing any RDM function. Upon entry to the exit program, register 1 points to an address list of the RDM function request parameters (see the following figure).



SKIPDML is the return code for the exit. RDM sets SKIPDML to 0 before calling the exit. If the exit sets SKIPDML to 1, RDM skips processing of the RDM function. However, RDM still calls the CSVXAFTTR exit. If SKIPDML has any other value than 1, RDM processes the function and then calls the CSVXAFTTR exit.

If you let RDM process the function, you may not alter the operation field in the TIS-CONTROL-AREA. You may, however, modify any of the other parameters. You can use this exit to perform additional sign-on and sign-off security checks and to monitor the activity of a particular file or view.

Because RDM used previous positioning information, use care if you intend to change any of the parameters passed to RDM for positioning information.

The release tape provides a default CSVXBFOR exit. It sets SKIPDML to 0 and does not modify any other parameters. It is linked by default with RDM. The source of the default exit is in the MACLIB member CSVXBFOR.

The contents of the parameters depend on the RDM command. If the command is **GET**, **UPDATE**, **INSERT**, **DELETE**, **MARK**, or **RELEASE**, the contents are the following:

	Address
CONTROL	TIS-CONTROL-AREA (see the DSECT C\$VTISCN for the contents).
DATA	User view data area as described by the INCLUDE statement for the <i>view-name</i> .
SELECT	Time-date stamp (Do not modify).
LIST	User view attribute data.
SKIPDML	4-byte return code (binary fullword integer). 1—Skip the RDML function. Any other value—Do not skip the RDM function.

If the command is **SIGN-ON**, **SIGN-OFF**, **FORGET**, **COMMIT**, **RESET**, or **NO-OP**, the contents are the following:

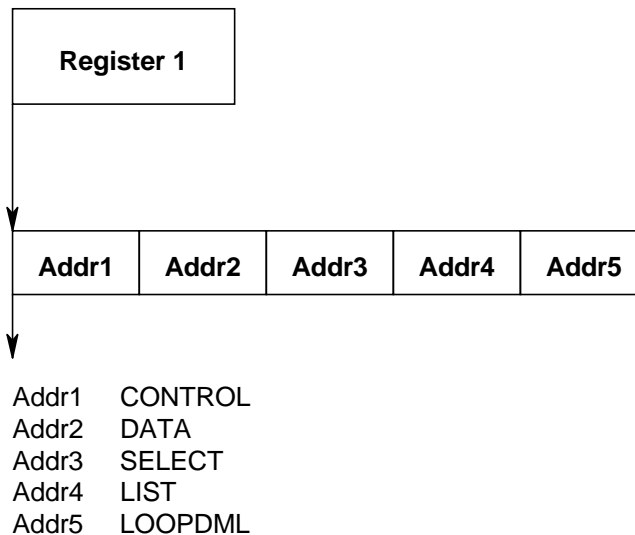
	Address
CONTROL	TIS-CONTROL-AREA (see the DSECT C\$VTISCN for the contents).
DATA	TIS-CONTROL-AREA.
SELECT	TIS-CONTROL-AREA.
LIST	TIS-CONTROL-AREA
SKIPDML	4-byte return code (binary fullword integer): 1—Skip the RDM function. Any other value—Do not skip the RDM function.

Using the after-function exit (CSVXAFTR)

Use the CSVXAFTR exit to analyze the return status from an RDM function call and to take appropriate action when an unsuccessful status is returned. Upon entry, register 1 points to an address list of the RDM function request parameters (see the following figure) which contains the values set by the CSVXBFOR exit and RDM's processing of the request.

LOOPDML is the return code for the exit. RDM sets LOOPDML to 0 before calling the exit. If the exit sets LOOPDML to 1, RDM repeats the processing of the RDML function. This processing includes calling the CSVXBFOR exit again. If LOOPDML has any value other than 1, RDM returns to the program that issued the RDM function.

The release tape provides a default CSVXAFTR exit. It sets LOOPDML to 0 and does not modify any other parameter. It is linked by default with RDM. The source of the default exit is in the MACLIB member CSVXAFTR.



The contents of the parameters depend on the RDM command. If the command is **GET**, **UPDATE**, **INSERT**, **DELETE**, **MARK**, or **RELEASE**, the contents are the following:

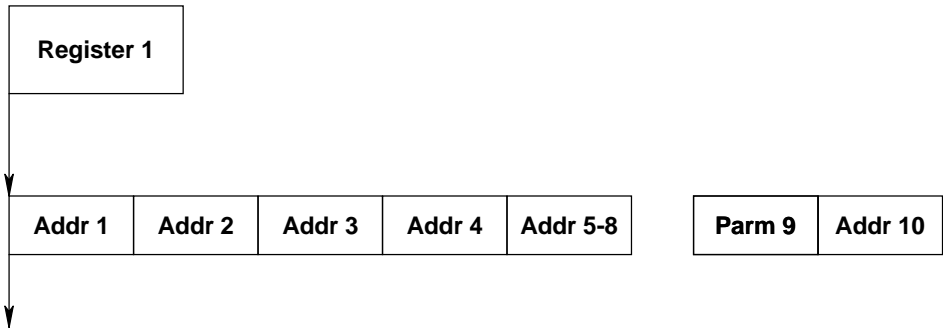
	Address
CONTROL	TIS-CONTROL-AREA (see the DSECT C\$VTISCN for the contents).
DATA	User view data area as described by the INCLUDE statement for the <i>view-name</i> .
SELECT	Time-date stamp (Do not modify).
LIST	User view attribute data.
LOOPDML	4-byte return code (binary fullword integer): 1—Skip the RDM function. Any other value—Do not skip the RDM function.

If the RDM command is **SIGN-ON**, **SIGN-OFF**, **FORGET**, **COMMIT**, **RESET**, or **NO-OP**, the contents are the following:

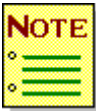
	Address
CONTROL	TIS-CONTROL-AREA (see the DSECT C\$VTISCN for the contents).
DATA	TIS-CONTROL-AREA.
SELECT	TIS-CONTROL-AREA.
LIST	TIS-CONTROL-AREA.
LOOPDML	4-byte return code (binary fullword integer): 1—Execute this RDM function again. Any other value—Continue processing and return to the program that issued the RDM function.

Using the TASKID exit (CSVXTSID)

Use the CSVXTSID exit to analyze and modify, or bypass a task ID which is attempting to access RDM. RDM invokes CSVXTSID before performing any RDM function. Upon entry to the exit program, register 1 points to an address list of the RDM function request parameters. The return code from this exit is not tested by RDM.



- Addr 1 Reserved for future use. Do not modify.
- Addr 2 Reserved for future use. Do not modify.
- Addr 3 Address of the CICS DFHEIB (EXEC Interface) control block for this task. Do not modify.
- Addr 4 Address of the CICS DFHEISTG (EXEC Interface Storage) control block for this task. Do not modify.
- Addr 5-8 Reserved for future use. Do not modify.
- Parameter 9 Task ID in the following format:
 - byte 1 = T (terminal task)
 - bytes 2-5 = Contents of the EIBTRMID field
 - bytes 6-8 = Reserved
- Addr 10 Reserved for future use. Do not modify.



This exit is called only when processing a terminal task.

The CSVXTSID exit is called via a BALR R14, R15 instruction. General purpose register contents are:

R0	Unpredictable
R1	Parameter list
R3-12	Unpredictable
R13	Standard 18 fullword save area
R14	Return address in RDM
R15	Entry point of CSVXTSID

You must restore registers to their original contents before returning control to RDM.

Using validation exits

RDM supports validation exits that allow the DBA to write more complex validation logic than is available using range checking or validation tables. See “[Validation options](#)” on page 41 for information on how RDM performs validation checking such as range, table, and validation exits.

You define a validation exit in the Directory using the Physical Field entity. Refer to the [SUPRA Server PDM Directory Online User’s Guide \(OS/390 & VSE\)](#), P26-1260, or the [SUPRA Server PDM Directory Batch User’s Guide \(OS/390 & VSE\)](#), P26-1261, for information on defining validation options for a physical field. To specify the validation exit, enter *E* in the VALIDATION OPTION field of the Physical Field entity. Specify the exit name in the VALIDATION EXIT field.

See the example listings at the end of this section for sample code used to switch to 24-bit or 31-bit mode.

For more information on the addressing mode for RDML exits, see “[Using RDML exits](#)” on page 249. You can specify many different validation exit names in the Directory. However, for a particular environment, RDM collects all validation exits into one validation exit module. The validation exit module names and register conventions are:

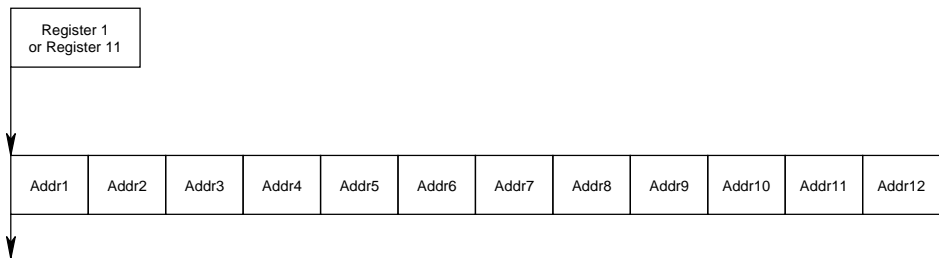
Module name	OS/390 & VSE batch CSVXIVXT	OS/390 & VSE CICS CSVXCVXT
Register 1	Parameter list	Parameter list
Register 11	-	-
Register 12	-	EIB
Register 13	-	EISTG
Register 14	Return address	Exit module entry
Register 15	Exit module entry point	-

The batch validation exit module, CSVXIVXT, is a separate load module. It is loaded dynamically the first time a view is opened that uses a physical field that specifies a validation exit. No register save area is provided. RDM saves and restores its registers. The return address is provided in register 14.

You can link the CICS Validation Exit module with the CSVNPLVS load module by adding an INCLUDE to the CSVNPLVS link deck for your exit program. It will be called by BALR 14,15 instruction and receive control in 31-bit mode.

Alternatively, the CICS Validation Exit module can be a separate load module. This requires that you add an entry for the exit program to the PPT. In this case, the Validation Exit receives control in 24-bit addressing mode. Normal CICS linkage conventions apply. The exit module must return by using a CICS RETURN statement.

Sample validation exit modules are supplied with RDM. Use them as starting points for developing your own exits. The parameter list contains twelve addresses, as follows:



1. Address of a 256-byte work area available to the exit. This is the same work area provided to the environment-dependent database exits. The contents of the work area remain unchanged from one call of an exit to the next. However, an RDML **SIGN-ON** or **SIGN-OFF** command initializes the work area to binary zeroes. The address of the work area may change from one exit call to the next; therefore, do not store any address that points to the work area.
2. Address of the return code. The return code is a 4-byte binary integer. The values the validation exit sets are:
 - 1 Invalid or unsupported validation exit name
 - 0 Valid value or valid exit name
 - 1 Invalid value
3. Address of the 8-character exit name. This is the exit name specified in the Physical Field entity on the Directory. If the exit name is invalid or not supported, the exit module sets the return code to -1.
4. Address of the 30-character user name. The user name provided on the RDML **SIGN-ON** command.

5. Address of the 30-character view name. The name of the view that contains the column being validated.
6. Address of the 30-character column name.
7. Address of the value to be validated. The length and format of this field varies.
8. Address of the 1-character type of the value:
 - C Character
 - P Packed
 - Z Zoned
 - B Binary
 - F Floating Point
 - K Kanji
9. Address of the length of the value. The length is a 4-byte binary integer.
10. Address of the 1-character signed flag for the value:
 - Y The value is signed.
 - N The value is not signed.
11. Address of the number of decimals in the value. The number is a 4-byte binary integer.
12. Address of the 1-character operation type. This field indicates the type of RDML request that caused the call of the validation exit:
 - G GET RDML
 - I INSERT RDML
 - U UPDATE RDML
 - O Open of the view. A value is not passed. Exit module should only validate the exit name. The validation exit should not change any parameters except for the return code. RDM calls the validation exit for each column in the view that requires a validation exit. For update requests, only columns that have been changed are validated.

When using derived views, the first view to see the data calls the validation process. That is, for GET RDML base view processing calls the validation process. For **UPDATE** and **GET**, the derived view processing calls the validation process.

Whenever RDM opens a view requiring validation exits, RDM calls the validation exit module to verify that it supports the exit name. If the validation exit module cannot support the exit name, it should set a -1 return code. If during an **OPEN**, RDM receives a -1 return code or it cannot load or find the validation exit module, RDM returns a function status indicator (FSI) indicating a fatal error and a message indicating the exit name for a column is invalid. If CSVXCVXT is link edited with CSVNPLVS, the Validation Exit receives control in 31-bit mode. Your exit code must switch to 24-bit mode if it issues commands that require 24-bit addressing capability. If your exit code switches to 24-bit mode, it must switch back to 31-bit mode before returning to RDM.

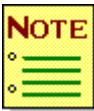
If CSVXCVXT is not linked with CSVNPLVS, it receives control in 24-bit mode via a CICS command level LINK. Your exit code must switch to 31-bit mode if it issues commands that require 31-bit addressing capability. If your exit code switches to 31-bit mode, it must switch back to 24-bit mode before returning to RDM.

The following code listing shows an example of code to switch to 24-bit addressing mode and then back to the original addressing mode. To switch from 24-bit mode to 31-bit mode and back, see the sample code shown in the next code listing.

```
*****
                                CODE TO SWITCH TO AMODE-24
*****
                                *****
                                LA      R9,LABELY      SETUP FOR RETURN TO ORIGINAL
                                LA      R14,LABELX      SETUP FOR SWITCH TO AMODE-24
                                BSM     R9,R14          CHANGE TO AMODE-24, SAVE AMODE
                                DC      H'0'          (NEVER WILL BE EXECUTED)
                                LABELX DS      0H
                                Your existing code which must execute in AMODE-24 is here.
                                *****
                                CODE TO RESTORE AMODE
                                *****
                                BSM     0,R9          RESTORE AMODE
                                DC      H'0'          (NEVER WILL BE EXECUTED)
                                LABELY DS      0H
```

The following code listing shows an example of code to switch from the current addressing mode to 31-bit mode, and then to restore to the original addressing mode. To switch to 24-bit mode and back, see the sample code shown in the preceding listing.

```
BEGIN          DS      0H
*
      STM      R14,R12,12(R13)    SAVE THE CALLERS REGISTERS
      L        R10,LABEL1          ESTABLISH PERMANENT BASE,
*                                  SET AMODE 31 BIT
      LA       R12,LABEL2          GET ADDRESS OF EXIT POINT
      BSM      R12,R10             SAVE CALLER'S AMODE, SET TO 31
      DS       0F                  ALIGNMENT
LABEL1        DC      A(BEGIN31+X'80000000')
BEGIN31       DS      0H
*****        NOW IN 31 BIT MODE SO WE CAN MANIPULATE EXTENDED CSA
      ST       R12,SAVEMODE        SAVE MODE REGISTER
... code that runs in 31-bit mode goes here ...
RETURN        DS      0H
      L        R12,SAVEMODE        RESTORE MODE REGISTER
      L        R13,4(,R13)         RESTORE ORIGINAL SAVE AREA
      BSM      0,R12              RESET CALLER'S AMODE
LABEL2        DS      0H
      L        R14,12(,R13)        RESTORE RETURN ADDRESS
      LM       R0,R12,20(R13)      RESTORE CALLER'S REGISTERS
      BR       R14                 RETURN WITH RC IN R15
SAVEMODE       DS      F           MODE REGISTER SAVE AREA
```



Although the parameters in the parameter list passed to the user exits are in 24-bit addressable storage, the parameter list may contain addresses that are in 31-bit storage.

B

Setting the online RDM options with macros

Overview of setting the online RDM options with macros

Set or revise your online RDM options for OS/390 or VSE with the C\$VOOPTM macro. Choose the parameter values for the options you want. Your SUPRA Server macro library contains the options module CSVOOPTM (OS/390) or CSVDOPTM.A (VSE) which invokes the macro C\$VOOPTM. Code the parameters you choose in the options module CSVOOPTM or CSVDOPTM.A (or a copy). Assemble and link the options module as the module CSVOOPTM. Online RDM loads the module. For VSE, the Assembler output is CSVDOPTM.OBJ.

Your SUPRA Server job control language (JCL) library contains a sample member named TXJVOPTM for assembling and linking CSVOOPTM. Samples are subject to change. See the SUPRA Server JCL library member TXJ\$INDX for a list showing this and other JCL samples. See the SUPRA Server Macro library member TX\$\$INDX for a list of different kinds of samples (UCLCODE, JCL, etc.). Refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250, for more information on JCL samples.

C\$VOOPTM	$\left[\text{ASMDATE} = \left\{ \frac{\text{'93.09.01'}}{\text{'yy.mm.dd'}} \right\} \right]$	X
	$\left[,\text{ASMTIME} = \left\{ \frac{\text{'23.59.59'}}{\text{'hh.mm.ss'}} \right\} \right]$	X
	$\left[,\text{BLKSIZE} = \left\{ \frac{\text{4K}}{\text{nnnK}} \right\} \right]$	X
	$\left[,\text{CICS} = \left\{ \frac{\text{NO}}{\text{YES}} \right\} \right]$	X
	$\left[,\text{CTXNS} = \left\{ \frac{\text{5}}{\text{nnnnn}} \right\} \right]$	X
	$\left[,\text{GETMAIN} = \left\{ \frac{\text{ANY}}{\text{BELOW}} \right\} \right]$	X
	$\left[,\text{GLOBSIZ} = \left\{ \frac{\text{2M}}{\frac{\text{nnM}}{\text{nnnnnK}}} \right\} \right]$	X
	$\left[,\text{HEAP\#} = \left\{ \frac{\text{5}}{\text{nnn}} \right\} \right]$	X
	$\left[,\text{HEAPSZ} = \left\{ \frac{\text{32K}}{\text{nnnnK}} \right\} \right]$	X
	$\left[,\text{IMSRDMP} = \left\{ \frac{\text{241}}{\text{nnn}} \right\} \right]$	X
	$\left[,\text{IMSSMPL} = \left\{ \frac{\text{65536}}{\text{nnnnnnnn}} \right\} \right]$	X
	$\left[,\text{IMSSMPX} = \left\{ \frac{\text{10240000}}{\text{nnnnnnnn}} \right\} \right]$	X

$\left[,\text{IMSSPCO} = \left\{ \frac{\text{CSGOPTNS}}{\text{XXXXXXXX}} \right\} \right]$	X
$\left[,\text{IMSSPOL} = \left\{ \frac{241}{nnn} \right\} \right]$	X
$\left[,\text{PFILE} = \left\{ \frac{\text{LV00}}{\text{xxxx}} \right\} \right]$	X
$\left[,\text{RDMUSR\#} = \left\{ \frac{5}{nnnnn} \right\} \right]$	X
$\left[,\text{RPTSIZE} = \left\{ \frac{32\text{K}}{nn\text{K}} \right\} \right]$	X
$\left[,\text{STACKSZ} = \left\{ \frac{16\text{K}}{nn\text{K}} \right\} \right]$	X
$\left[,\text{SYNCTYP} = \left\{ \frac{\text{Y}}{\text{N}} \right\} \right]$	X
$\left[,\text{SYSTEM} = \left\{ \frac{\text{OS}}{\text{DOS}} \right\} \right]$	X
$\left[,\text{TCISIZE} = \left\{ \frac{32\text{K}}{nn\text{K}} \right\} \right]$	X
$\left[,\text{TSLVP} = \left\{ \frac{\text{PLVS}}{\text{xxxx}} \right\} \right]$	X
$\left[,\text{TSROLL} = \left\{ \frac{\text{A}}{\text{M}} \right\} \right]$	X

ASMDATE = $\left\{ \begin{array}{l} \text{'93.09.01'} \\ \text{'yy.mm.dd'} \end{array} \right\}$

Description *Optional.* Specifies the Assembly Date stamp for the generated options table.

Default '93.09.01'

Format 6 numeric characters delimited by periods and enclosed in single quotation marks.

Considerations

- ◆ OS/390 users can code ASMDATE = &SYSDATE to obtain the current system date.
- ◆ VSE users must code the current date value manually if they wish to accurately timestamp the resultant options table generation. The VSE Assembler does not support the global value &SYSDATE.

,ASMTIME = $\left\{ \begin{array}{l} \text{'23.59.59'} \\ \text{'hh.mm.ss'} \end{array} \right\}$

Description *Optional.* Specifies the Assembly Time stamp for the generated options table.

Default '23.59.59'

Format 6 numeric characters delimited by periods and enclosed in single quotation marks.

Considerations

- ◆ OS/390 users can code ASMTIME = &SYSTIME to obtain the current system time.
- ◆ VSE users must code the current time value manually if they wish to accurately timestamp the resultant options table generation. The VSE Assembler does not support the global value &SYSTIME.

,BLKSIZE = $\left\{ \begin{array}{l} 4K \\ nnnK \end{array} \right\}$

Description *Optional.* Indicates the block size (physical record size) of the heap roll area.

Default 4K

Options 4K–28K

Considerations

- ◆ The block size must be a multiple of 4K.
 - ◆ The block size should be as close as possible to the maximum track size of your physical device without exceeding it. For IBM 3390 disk devices, Cincom recommends a block size of 28K.
 - ◆ The heap size is rounded up to be a multiple of the block size.
-

,CICS = $\left\{ \begin{array}{l} NO \\ YES \end{array} \right\}$

Description *Optional.* Indicates whether RDM is running under CICS.

Default NO

Considerations

- ◆ If you are running under CICS, you must code CICS=YES.
 - ◆ If you are running under IMS/DC, you must code CICS=NO or let it default to NO.
-

,CTNXNS = $\left\{ \begin{array}{l} 5 \\ nnnnn \end{array} \right\}$

Restriction This parameter is ignored unless you are running SPECTRA under CICS.

Description *Optional.* Specifies the number of heaps for use by SPECTRA.

Default 5

Options 1–32767

,GETMAIN= $\left\{ \begin{array}{l} \text{ANY} \\ \text{BELOW} \end{array} \right\}$

Description	<i>Optional.</i> Specifies whether to use memory below the 16 MB line for RDM heaps and global views.	
Default	ANY	
Options	ANY	Allocate RDM heaps and global views using the LOC=ANY option of the IBM macro GETMAIN. In practice, this implies they are allocated above the 16 MB line.
	BELOW	Allocate RDM heaps and global views below the 16 MB line.

,GLOBSIZ= $\left\{ \begin{array}{l} \text{2M} \\ \text{nnM} \\ \text{nnnnnK} \end{array} \right\}$

Description	<i>Optional.</i> Specifies the size of the memory area for storing global views.	
Default	2M	
Format	1–5 numeric characters followed by K, or 1–2 numeric characters followed by M	

Considerations

- ◆ Under CICS, if you set this parameter to zero, RDM will allocate NO global view memory and WILL NOT initialize a global view area. If you want to have a global view area, you MUST enter a non-zero value for this parameter or leave the default value of 2 MB.
- ◆ If you allocate global views below the 16 MB line by coding GETMAIN=BELOW, you must specify a global view area size that fits below the line in your address space. The default size will almost certainly be too large.

,HEAP# = $\left\{ \begin{array}{l} 5 \\ nnnnn \end{array} \right\}$

Description *Optional.* Specifies the number of heaps to be allocated.

Default 5

Options 1–32767

Considerations

- ◆ HEAP# controls the concurrency of tasks. RDMUSR# controls the maximum number of tasks allowed to sign on to RDM.
- ◆ If HEAP# is less than RDMUSR#, rolling of pseudoconversational task context will occur when the number of tasks signed on to RDM exceeds HEAP#.
- ◆ If HEAP# is equal to RDMUSR#, no rolling of pseudoconversational task context will occur.
- ◆ If HEAP# is greater than RDMUSR#, you are wasting space. In this situation, heaps will be allocated but never used, because RDMUSR# limits the number of tasks allowed to sign on to RDM.

,HEAPSZ = $\left\{ \begin{array}{l} 32K \\ nnnnK \end{array} \right\}$

Description *Optional.* Specifies the size of the heaps to be allocated.

Default 32K The default options module distributed with SUPRA Server specifies a HEAPSZ of 64K.

Options 4K–1020K

Considerations

- ◆ This size must be at least as large as the size specified by the BLKSIZE parameter.
- ◆ The heap size is rounded up to be a multiple of the block size.

,IMSRDMP = $\left\{ \begin{array}{l} \underline{241} \\ nnn \end{array} \right\}$

Description *Optional.* Specifies the number of the memory subpool where RDM work space is allocated.

Default 241

Format 3 numeric characters

Considerations

- ◆ If you are running under CICS, Cincom recommends you code IMSRDMP=0.
- ◆ The memory subpool number specified by this parameter must be the same as that specified by the IMSSPOL parameter.

,IMSSMPL = $\left\{ \begin{array}{l} \underline{65536} \\ nnnnnnn \end{array} \right\}$

Restriction This parameter is ignored unless you are using SPECTRA under CICS.

Description *Optional.* Specifies the standard memory pool size, in bytes, for SPECTRA system memory.

Default 65536

Format 1–7 numeric characters

,IMSSMPX = $\left\{ \begin{array}{l} \underline{10240000} \\ nnnnnnn \end{array} \right\}$

Restriction This parameter is ignored unless you are using SPECTRA under CICS.

Description *Optional.* Specifies the maximum total amount of memory, in bytes, allowed for SPECTRA system memory.

Default 1024000

Format 7 numeric characters

Consideration

- ◆ Cincom recommends you allow this value to default to 1024000.

```
,IMSSPCO = {CSGOPTNS
             {XXXXXXXX}}
```

Restriction This parameter is ignored unless you are using SPECTRA under CICS.

Description *Optional.* Specifies the name of the SPECTRA options module (assembled using the macro CSGOPTNS).

Default CSGOPTNS

Format 1–8 alphanumeric characters or @, #, \$; first character must be alphabetic or @, #, \$

```
,IMSSPOL = {241
            {nnn}}
```

Description *Optional.* Specifies the number of the memory subpool where SPECTRA work space is allocated.

Default 241

Format 1–3 numeric characters

Considerations

- ◆ Cincom recommends you always code IMSSPOL=0.
- ◆ This field in the control block is accessed only by the memory manager utility CSVOCSA.
- ◆ The memory subpool number specified by this parameter must be the same as that specified by the IMSRDMP parameter.

,PFILE = $\left\{ \begin{array}{l} \text{LV00} \\ \text{xxxx} \end{array} \right\}$

Restriction This parameter is ignored unless you are using SPECTRA under CICS.

Description *Optional.* Specifies the name of the SPECTRA Personal File.

Default LV00

Format 4 alphanumeric characters or @, #, \$; first character must be alphabetic or @, #, \$

,RDMUSR# = $\left\{ \begin{array}{l} 5 \\ \text{nnnnn} \end{array} \right\}$

Restriction This parameter is ignored under IMS/DC.

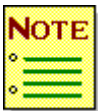
Description *Optional.* Specifies the maximum number of concurrent active RDM users.

Default 5

Format 1–5 numeric characters (1–32767)

Considerations

- ◆ If RDMUSR# exceeds HEAP#, heap storage is rolled to temporary auxiliary storage (for pseudoconversational tasks).
- ◆ If the number of users attempting to use RDM exceeds RDMUSR#, functional status indicators of “I” (insufficient resources) will be returned.
- ◆ RDMUSR# is used internally by RDM CICS XA support to build a table which determines how many heaps can be accessed.
- ◆ RDM allocates a stack only for the life of an RDML command (a stack will be allocated at the beginning of an RDML command and freed at the end of that command).



See also the considerations listed under the HEAP# parameter in this section, since these two parameters work in unison.

,RPTSIZE = $\left\{ \begin{array}{l} \underline{32K} \\ nnK \end{array} \right\}$

Description *Optional.* Indicates the amount of memory in the RPT table to be used for DBAID utility requests.

Options 4K–31K

,STACKSZ = $\left\{ \begin{array}{l} \underline{16K} \\ nnK \end{array} \right\}$

Description *Optional.* Specifies the size of the stacks to be allocated.

Default 16K

Format 1–2 numeric characters followed by K

Considerations

- ◆ This size must not be smaller than 16K or larger than 63K.
 - ◆ One heap and one stack comprise one slot.
-

,SYNCTYP = $\left\{ \begin{array}{l} \underline{Y} \\ N \\ R \end{array} \right\}$

Description *Optional.* Specifies the type of sync point performed by RDM.

Options Y Full sync point/rollback

 N No sync pointing

 R Commit sync point; no rollback

,SYSTEM = $\left\{ \begin{array}{l} \underline{OS} \\ DOS \end{array} \right\}$

Description *Required.* Specifies the operating system type in which RDM will be used.

Default OS

Options OS RDM is used in an OS/390/XA or OS/390/ESA system.

 DOS RDM is used in a VSE/SP or VSE/ESA system.

,TCISIZE = $\left\{ \begin{array}{l} 32K \\ nnK \end{array} \right\}$

Description *Optional.* Specifies the Control Interval size for the temporary storage file on auxiliary storage.

Format 1–2 numeric characters followed by K

Consideration This value must be the same as the Control Interval size you specify for your VSAM auxiliary file.

,TSLVP = $\left\{ \begin{array}{l} PLVS \\ xxxx \end{array} \right\}$

Restriction This parameter is ignored except under CICS.

Description *Optional.* Specifies the prefix of the CICS temporary storage identifier.

Default PLVS

Format 4 alphanumeric characters or @, #, \$; first character must be alphabetic or @, #, \$

Consideration

- ◆ Cincom recommends you allow this value to default to PLVS.
-

,TSROLL = $\left\{ \begin{array}{l} A \\ M \end{array} \right\}$

Description *Optional.* Specifies the type of temporary storage destination for RDM heap rollout/roll-in.

Options A Roll heaps to auxiliary storage

 M Roll heaps to virtual main storage above the 16 MB line

Index

A

- Access
 - Definition 72
 - Definitions 64
 - keyed 33
 - order of 64
- Access Key, and logical key 33
- Access methods, impact of
 - positional relationships 55
- Access set, and view binding 125
- ACCESS statement, in view
 - access definition 32, 72
- After-function exit, RDML 252
- ALL keyword, in ALLOW clause 49
- ALL parameter, for DBAID
 - commands
 - in BIND command 140
 - in delete command 153
 - in UNDEFINE command 203
- ALLOW clause
 - for maintenance 49
 - in access definition 75
- Environment description 122
- Application program, RDM
 - reporting on 223
 - statistics gathering in 129
- Application programmers
 - and RDM 19
 - and views 25
- Application tasks, types
 - supported 128
- Application, RDM
 - currentness of 115
 - linking and executing 226
- ASI. *See* Automatic System Initialization and Column Status Indicator (ASI)
- AT clause, in DBAID commands
 - in GET command 162
 - in MARK command 178

- AT CLAUSE, in DBAID
 - commands
 - in GO command 164
- Automatic
 - COMMIT 199
 - disabling 145
 - hold 162
- Automatic RESET, disabling 172
- Automatic System Initialization (ASI) 127

B

- Base relations, examples of 76
- Base view 20, 24, 29
 - creating 29
 - DBAID 63
 - DBAID and 24
 - DBAID sample session 106
 - defining 29
 - Directory Maintenance and 25, 63
 - examples 78
 - generating with Directory Maintenance 63
 - global 122
 - integrity and 87
 - opening 85
 - relating to users 130
 - sources for derived views 64
- Batch RDM, and global views 122
- Before-function exit, RDML 250
- BIND command
 - an COMMIT command 149
 - and RESET command 192
 - and view binding 125
 - example 141
 - syntax 140
- BIND parameter, in SAVE
 - command 193
- Binding views 125
- Blank, as validation option 40
- BOUND parameter, in BIND
 - command 140
- Bound views 125
- Built-in view commands, in
 - DBAID 138
- BYE command
 - and SIGN-OFF command 197
 - and STATS-OFF command 200
 - and STATS-ON command 201
 - and view binding 125
 - syntax 142

C

- C\$VOOPTM macro 261
 - format 261
 - parameters 264
- Cascade delete 53
 - and integrity 97
- CAUTIOUS command
 - and COMMIT command 149
 - syntax 145
- Change
 - logical 115
 - physical 115
 - to files 115
- Characters per line, displaying 175
- CICS
 - and global views 122
 - and recovery 132
 - processing 236
- Cincom Software Selection
 - Screen 105
- COBOL Programmers' Report 215
- Column
 - defining 64
 - redundant
 - in view column definition 68
 - joins and 43
 - required 31
 - required, designating 39
- Column definition
 - examples 71
 - syntax 64
- Column description, displaying 146, 158
- Column list, displaying 207
- Column name
 - displaying 143
 - in COLUMN-DEFN command 146
 - in COLUMN-TEXT command 146
 - in view column definition 67
 - uniqueness 66
- Column parameter
 - in OPEN command 180
 - in UPDATE command 204
- Column Status Indicator (ASI) 58
- Column values, shared 54
- COLUMN-DEFN command
 - example 147
 - syntax 146
- Commands
 - built-in view commands 138
 - DBAID 133
 - editing commands 136
 - example 148
 - format 135
 - list 136
 - reissuing 139
 - Relational Data Manipulation language (RDML)
 - commands 137
 - statistics commands 138
 - syntax 148
 - system commands 136
- COMMIT, DBAID command 132
- Commits
 - automatic 199
 - BIND command 141
 - CAUTIOUS command 145
 - DENY command 155
 - PERMIT command 183
 - REMOVE command 189
 - RESET command 192
 - SAVE command 193
- Compound nonunique key 38
- Compound unique key 37
- Concatenated key 37
- Conceptual schema 24
- CONST keyword 39
- CONST option, in column
 - definition 39, 66
- Context file, RDM 225
- Control-key, on PDM file 33
- COPY command
 - example 151
 - syntax 150
- Core-image library 129
- CSVLVRES. *See* RDM, modules
- CSNVRES. *See* RDM, modules
- Currentness of program 115
- Currentness of view bindings, checking 120

D

Data retrieval, with RDM 55

Data storage areas 226

Data validation, automatic 19

Database

changing contents of 49

IMS

RDM access 23

navigation 55

penetration 56

sample of 50

updating 201

Database user exits, RDM 240

DBA (Database Administrator),
function in RDM 27

DBA Report

and view binding 126

described 213

DBA, function in RDM 27

DBAID utility

base views 24

command categories 134

command format 134

commands

built-in view commands 138

non-DBA users 104

RDML 137

statistics 138

system 136

commands list 136

DBA and 27

editing commands 136

examples, online 105

positional parameters 133

signing on 105

uses for 24, 104

view definition with 63

DECLARE clause, on RDM

Programmers' Report 215

Default validation 42

Default values 20

for physical fields 47

DEFINE

and EDIT command 156

and LINE-NUMBER command
174

and LIST command 177

and SAVE command 193

and UNDEFINE command 203

in sample DBAID session 111

syntax 152

Delete

cascade delete 97

column status indicators (ASIs)
60

integrity 89

nullify delete 97

nulls and 45

restrict delete 97

DELETE command

example 154

syntax 153

DENY command 130

and COMMIT command 149

and RESET command 192

syntax 155

Derived view 20, 29

application programmer and 26

building 29

customizing 31

defining 64

defining in sample DBAID
session 111

examples of 80

external schema and 25

global 122

maintenance restrictions 31

prototyping 25

relating to users 130

required columns in 70

size recommended 31

sources of derived views 64

SPECTRA users 25

tailoring 64

testing with DBAID 25, 31, 104

Derived views

processing 85

Directory Maintenance

and base views 24

for defining global views 122

for view definition 63

Directory reports 211

Directory, and RDM 27, 41

DMLPRINT file 184

Domain checking, overriding 67

Domain information, and

NORMAL 40

Domains 40

DTB (Dynamic Transaction

Backout) 132

E**EDIT**

- and DEFINE command 152
- and LINE-NUMBER command 174

- and LIST command 177
- and SAVE command 193
- and UNDEFINE command 203

Editing commands, in DBAID 136

End User Report 218

- and view binding 126

END. keyword, with MASS
parameter 169

Environment description
parameters 230

Equal sign, in domain checking
43

ERASE command, syntax 157

Exit

- after-function 252
- before-function 250
- database exits 238
- function exit
 - environment-dependent 245
 - environment-independent 240
- processing flow 237
- purpose 237
- RDML exits 249
- status exit
 - environment-dependent 245
 - environment-independent 242
- task ID 254
- types 237
- validation exits 256
 - option specifying 40

Exit from DBAID 142

- in sample DBAID session 114

External schema 25

F**FIELD-DEFN**

- syntax 158

FIELD-DEFN command

- example 158

FIELD-TEXT command 148

File changes 117

File structures 19, 24

- and external schema 25
- physical changes report 220
- RDM context 225

FIRST parameter

- in GET command 161
- in GO command 164
- in INSERT command 168

FKEY

- and derived views 90
- and insertion integrity 90
- and update integrity 90

FOR parameter, in GO command
164

FOR UPDATE parameter, in GET
command 162

Foreign key 52

- and deletion integrity 96
- defined 87
- defining 90
- nullifying 96
- redundant, with GET

- processing 95

- value integrity 90

FORGET

- syntax 160

FORGET, and MARK command
178

FROM parameter, in GO
command 164

FSI. *See* Function Status
Indicator (FSI)

Function exit 243, 248

- in sample DBAID session 105

Function Status Indicator (FSI)
59

- for foreign key on rejected
insert 91

- for foreign key on rejected
update 93

G

GENERATE, and base views 29

GET command

- and DELETE command 153
- and UPDATE command 205
- examples 161
- in sample DBAID session 111
- syntax 161

GET processing

- and integrity 95
- and null values 44
- and validation 44

GETVIS area, RDM loaded in
126

GIVING clause, in Access
 Definition 74
 Global view area 122
 definition and example 122
 in memory 123, 228, 266
 performance optimization 122
 GO command 137
 examples 167
 in sample DBAID session 110,
 114
 syntax 164

H

Hardware requirements for RDM
 28
 Heaps
 allocation of 228
 defined 226
 number, specifying 265, 267
 size, specifying 265

I

Impact of Change Report 220
 IMS database
 RDM access 23
 recovery and 132
 IMS/DC, global view support 122
 Index
 and unbound views 117
 as access method 55, 57
 deletion integrity 96
 INSERT command 137
 automatic 51
 examples 170
 in DBAID session 112
 reject 51
 syntax 168
 INSERT processing 86
 and null values 45
 and validation 45
 integrity 91
 Integrity 25
 and base views 29
 and cascade delete 97
 and foreign key value 89, 90
 and GET processing 95
 and nullify delete 97
 and restrict delete 97

centralizing 31
 database 19, 24
 delete 53, 89, 96
 example used in ACCESS
 statement 51
 examples of 97
 insertion 91
 maintaining 87
 rules for 89
 update 93
 Internal schema 23

J

JCL samples, RDM in SUPRA
 Server libraries 134, 211,
 225, 248, 261
 Join compatibility 43

K

KEEP command, syntax 172
 Key
 access with 33
 compound nonunique 38
 compound unique 37
 concatenated 37
 constant 33
 control-key, PDM file 33
 nonunique 33, 38
 primary
 defined 87
 integrity 91
 simple nonunique 38
 simple unique 36
 unique 33
 Key, qualifier in view definition 65
 Keyed access 33
 KSDS VSAM files 23
 and recovery 132

L

LAST parameter
 in GET command 161
 in GO command 164
 in INSERT command 168
 Level of occurrence, and BY-
 LEVEL command 143

- Line-number command
 - and DEFINE command 152
 - and EDIT command 156
 - and LIST command 174
 - example 174
 - syntax 173
- Line-number* parameter, in line-number command 173
- LINESIZE command
 - syntax 175
- Link decks
 - for RDM interface 248
 - for RDM resident module 126
 - RDM samples in SUPRA
 - Server libraries 248
- Linkpath area (LPA)
 - for performance optimization 126
 - installing RDM in 126
- Linkpaths, unbound views 116
- LIST
 - and DEFINE command 152
 - and EDIT command 156
 - and LINE-NUMBER command 174
 - and OPEN command 181
 - and REMOVE command 189
 - and SAVE command 193
 - and UNDEFINE command 203
 - syntax 176
- Lock 162
- Logical design, changing 116
- Logical key
 - and access key 33
 - example 33
 - number in view 33
 - order of 64
 - with fixed values 39
- Logical unit of work 149

M

- Macro, C\$VOOPTM 261
- Maintenance restrictions, and derived views 31
- MARK command
 - and FORGET command 160
 - and RELEASE command 188
 - syntax 178
- Mark-name* parameter, in FORGET command 160

- MARKS command
 - example 179
 - syntax 179
- MASS parameter, in INSERT command 169
- Memory
 - DBAID requirements 181
 - DBAID, conserving 178
 - extended 126, 225, 266
 - freeing 160, 203
 - global views and 122, 266
 - shared 126
- Modifying a view definitions in sample DBAID session 114
- Modules, for RDM 227

N

- Navigation
 - boundaries 57
 - constraints 57
 - database 55
- NEXT parameter
 - for GET command 161
 - for GO command 164
 - for INSERT command 168
- Nonunique key 33, 38
- NONUNIQUE KEY, qualifier in view column definition 65
- NORMAL
 - base views and 24, 29
 - DBA and 27
 - domains and 40
- Null value(s) 19
 - column 44
 - COLUMN-DEFN command and 40
 - constant value and 70
 - GET processing and 95
 - insertion integrity and 87
 - integrity and 89
 - KEY qualifier for view column definition and 64
 - MASS parameter for DBAID commands and 169
 - update integrity and 93
 - validation of 42, 45
- Nullify delete 53
 - integrity and 97

Number-of-characters parameter,
for DBAID command
LINESIZE 175
Number-of-lines parameter, for
DBAID command
PAGESIZE 182

O

ONCE clause, in view access
definition 72
Online DBAID session, sample
105
OPEN
LIST command and 181
OPEN command
syntax 180
OPEN, DBAID command
in sample DBAID session 106
UNDEFINE command and 203
OPEN, DBAID command
example 181
OPER CONNECT
parameters 232
process 231
Operating systems
OS/390/ESA 225, 261
OS/390/XA 225, 261
OS/390/ESA 225, 261
OS/390/XA 225, 261

P

Packed decimal fields validation
42
PAGESIZE, DBAID command,
syntax 182
Password parameter, for DBAID
command SIGN-ON 198
Pause job 128
PDM. See Physical Data
Manager (PDM)
Penetration
access method 55
PERMIT
COMMIT command and 149
DBAID command 132
DENY command and 155
RESET command and 192
syntax 183

Physical Data Manager (PDM)
file(s)
RDM access 23
recovery 132
required for RDM 28
thread processing 235
Physical design, changing 117
Physical file, keyed access 33
Physical key 33, 35
Picture clause, on RDM
Programmers' Report 215
PL/1 Programmers' Report 215
Position, current, marking 178
Precompilers, RDML 21, 225
Primary key
defined 87
deletion integrity and 96
insertion integrity and 91
update integrity and 93
PRINT-STATS
DBAID command 128
example 184
STATS-ON command and 201
syntax 184
PRIOR parameter
for DBAID commands 134
for GET command 161
for GO command 164
for INSERT command 168
Procedure samples, RDM, in
SUPRA Server libraries
134, 211, 225
Processing time, displaying 205
Program changes, determining
impact of 115
Programmers' Report
described 215
Programmers' Report
view binding and 125
Public user 185
PUBLIC-DENY, DBAID
command
PUBLIC-PERMIT command
and 185
syntax 185
PUBLIC-PERMIT, DBAID
command
Directory Maintenance RELATE
function and 186
syntax 186

PUBLIC-VIEWS, DBAID
 command
 syntax 187
 VIEWS-FOR-USER command
 and 210

R

Range checking 41
RDM (Relational Data Manager)
 benefits of 19
 Directory and 40
 installing in linkpath area (LPA)
 126
 installing in shared virtual area
 (SVA) 126
 linking 261
 maintaining 113
 modules
 interface modules, linking
 exits with 247
 list 227
 resident module, moving to
 shared memory 126
 online options, OS/390,
 specifying 261
 reports, and stored views 29
 samples, in SUPRA Server
 libraries 211, 225, 248, 261
 security 25
RDML. See Relational Data
 Manipulation Language
 (RDML)
Rebinding views 115, 117, 120
Recompile 115, 120
Records
 recovery 132
Records, operations on. See Get,
 Insert, Update, and/or
 Delete
Redundant columns in view
 column definition 68
 joins and 43
Referential integrity examples 99
 maintaining 87
 rules 89
RELATE
 DBAID command PUBLIC-
 PERMIT and 186
 function of Directory
 Maintenance 132

Relational Data Manager. See
 RDM
Relational Data Manipulation
 Language (RDML) 19
 DBAID commands 137
 exits 249
 precompilers 21, 225
 processing 233
 verbs. See Get, Insert, Update,
 and/or Delete
Relational operators 19
Relationships
 creating 156
 removing 132, 185
RELEASE
 DBAID command 188
 in sample DBAID session 114
 LIST command and 177
 OPEN command and 181
 syntax 188
REMOVE
 COMMIT command and 149
 DBAID command 132
RENUMBER, DBAID command,
 syntax 191
Reports
 DBA and 27
 RDM 25, 211
 types of 211
REQ
 effect on processing 65
 foreign key and 90
 get processing and 95
 qualifier in view column
 definition 65
REQ qualifier and 39
Required column
 CONST qualifier and 39
 null value and 45
Required columns
 designating 34
RESET
 DBAID command 132
 syntax 192
RESET command
 view binding and 125
RESET command and
 syntax 192

Reset(s)
 automatic, disabling 172
 BIND command and 141
 DENY command 155
 ERASE command and 157
 PERMIT command and 183
 REMOVE command and 189
 SAVE command and 193
 Restart of task, during recovery 132
 Restrict delete 53
 integrity and 97
 Retrieval of rows (records)
 by penetration 164
 by sweeping 164
 Retrieval of rows (records). *See* Get
 Retrieval validation flag 41, 44
 Rollback 190
 during recovery 132
 Row(s)
 how RDM constructs 32
 operations on. *See* Get, Insert, Update, and/or Delete

S

SAME parameter
 for DBAID commands 134
 for GET command 161
 for GO command 164
 Samples, RDM, in SUPRA
 Server libraries 211, 225, 248, 261
 SAVE
 BYE command and 142
 COMMIT command and 149
 DBAID command 193
 EDIT command and 156
 examples 194
 LIST command and 177
 RESET command and 194
 syntax 193
 view binding and 193
 Schema name
 BIND command in 140
 COPY command and 150
 EDIT command, in 156
 LIST command 176
 LIST command, security in 176
 REMOVE command and 189
 SAVE command in 193

Schema(s)
 conceptual 24
 external 25
 internal 23
 Secondary key(s), unbound views and 116
 Security
 assigned on user basis 132
 base views and 29
 constant key and 39
 controlling 39
 DENY command and 155
 derived views and 39
 imposing on derived views 64
 of database 21
 PERMIT command and 183
 RDM 26
 Selection criteria, in view access definition 74
 Shared column values 54
 Shared virtual area (SVA)
 installing RDM in
 installation procedure 127
 optimizing performance 127
 SHOW-NAVIGATION
 DBAID command 195
 example 195
 syntax 195
 SIGN-OFF
 BYE command and 142
 DBAID command 197
 syntax 197
 SIGN-ON, DBAID command
 example 198
 syntax 198
 Simple nonunique key 38
 Simple unique key 36
 Slot(s)
 size
 global views and 124
 MARK command and 178
 specifying 267
 Slots
 defined 225
 Software Selection Screen 105
 Source file. *See* Source relation
 Source relation
 defined 87
 deletion integrity and 96
 integrity and 89
 nullify delete and 97
 update integrity and 93

SPECTRA

- user, views and 26
- views and 21

Stack(s)

- allocation of 228
- defined 225
- number, specifying 267
- size, specifying 267

Statistics

- BYE command and 142
- commands, in DBAID 138
- disabling 200
- displaying on terminal 199
- examples of 129
- gathering 128, 201
- interpreting 128
- printing 184, 201
- RDM 129
- SIGN-OFF command and 197

STATS

- DBAID command 128
- example 199
- STATS-ON command and 201
- syntax 199

STATS-OFF

- DBAID command 128
- PRINT-STATS command and 184
- STATS command and 200
- syntax 200, 201

STATS-ON

- DBAID command 128
- PRINT-STATS command and 184
- STATS command and 201
- STATS-OFF command and 201

Status

- indicators 58

Status exits 248**Status exits, RDM 244****Storage**

- configuring 228
- freeing 160, 188, 203

SURE, DBAID command, syntax 202**SVA. See Shared virtual area****Sweep**

- as access method 55, 57
- get requests 164

System commands, in DBAID 135**T****Table checking 42****Tabular data structure 19****Target file. See Target relation****Target relation**

- defined 87
- integrity and 89
- nullify delet and 97
- update integrity and 93

Task abend, recovery and 132**Task level recovery (TLR) 132**

- RESET command and 192

Task restart, during recovery 132**TASKID exit, RDML 254****Text, displaying for column 148****Thread, processing 235****Three-schema architecture 25****TIS-CONTROL-AREA, statistics gathering and 129****TIS-OPTIONS file, statistics gathering and 129****TLR. See Task level recovery****Transaction level recovery (TLR).**

- See Task level recovery

Two-schema architecture 24**U****UNDEFINE, DBAID command 203**

- DEFINE command and 152

- LIST command and 177

- syntax 203

- view binding and 125

Unique key 35**UPDATE**

- DBAID command 204

- example 204

- in sample DBAID session 113

- syntax 204

UPDATE keyword

- in view access definition 49, 52, 75

Update(s)

- anomalies, avoiding 31

- committing 149

- reject 52

User name parameter

- for DENY command 155

- for PERMIT command 183

User view(s)
 defined 21
 displaying 207
 memory required for 123
 name, parameter, for OPEN
 command 180
 User(s), number, specifying 270
 USER-LIST, DBAID command
 example 207
 syntax 207
 User-to-view relationships,
 security and 26
 USING parameter, for DBAID
 commands
 for GET command 163
 for GO commands 166

V

Validation
 criteria for derived view 31
 during get processing 44
 during insert processing 44, 51
 during update processing 44,
 51
 exits 256
 option specifying 41
 information, COLUMN-DEFN
 command and 40
 of defaults 42
 options 41
 packed and zoned decimal
 fields 42
 table, on the Directory 42
 Validity Status Indicator (VSI) 58
 in sample DBAID session 106
 Validity Status Indicators (VSI) 62
 VARIABLE EDIT, Directory
 Maintenance command, for
 view definition 63
 View definition(s) 64
 changing, in sample DBAID
 session 114
 removing 189
 storage of 25, 29
 testing with DBAID 29

text
 displaying for ****PUBLIC****
 user 187
 displaying for signed-on user
 187
 view binding 125
 with view binding 125
 View(s)
 access 21, 64
 and security 26
 active, displaying 209
 base. See Base view(s)
 binding, for performance
 optimization 125
 bound 115
 changes, determining impact of
 115
 closing 188
 definition. See View
 definition(s)
 derived. See Derived view(s)
 description, displaying 208
 displaying 168
 maintenance action for 21
 name(s), access definition, in
 72
 opening 180
 rebinding 115, 140, 141
 relating to user(s) 130, 183,
 186
 removing 189
 subsetting for user(s) 22
 testing 104
 unbound 115
 unnormalized, and nonunique
 keys 38
 user. See User view(s)
 virtual. See Virtual view(s)
 VIEW-DEFN, DBAID command
 example 208
 syntax 208
 Views
 Impacting Programs Report
 223
 Impacting Views Report 222
 Used by Programs Report 115,
 224
 VIEWS, DBAID command
 example 209
 syntax 209

VIEWS-FOR-USER, DBAID

command

example 210

syntax 210

Virtual view(s)

COPY command in 151

DEFINE command in 152

effect of BYE command on 142

modifying 156

removing 203

renumbering 191

VSI. See Validity Status Indicator
(VSI)

W

WHERE clause, in view access
definition 74

Z

Zoned decimal fields validation
42